
GemStone®

*System Administration Guide
for GemStone/S 64 Bit*

Version 2.2
for UNIX

April 2007

GEMSTONE[™]S 64

INTELLECTUAL PROPERTY OWNERSHIP

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

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

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

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

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

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

COPYRIGHTS

This software product, its documentation, and its user interface © 1986-2007 GemStone Systems, Inc. All rights reserved by GemStone Systems, Inc.

PATENTS

GemStone is covered by U.S. Patent Number 6,256,637 "Transactional virtual machine architecture", Patent Number 6,360,219 "Object queues with concurrent updating", and Patent Number 6,567,905 "Generational Garbage Collector". GemStone may also be covered by one or more pending United States patent applications.

TRADEMARKS

GemStone, GemBuilder, GemConnect, and the GemStone logos are trademarks or registered trademarks 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.

Sun, Sun Microsystems, Solaris, and SunOS are trademarks or registered trademarks of Sun Microsystems, Inc. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. SPARCstation is licensed exclusively to Sun Microsystems, Inc. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

HP and HP-UX are registered trademarks of Hewlett Packard Company.

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

Microsoft, MS, Windows, Windows 2000 and Windows XP 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.

AIX and POWER4 are trademarks or registered trademarks of International Business Machines Corporation.

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. All terms mentioned in this documentation that are known to be trademarks or service marks have been appropriately capitalized to the best of our knowledge; however, GemStone 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.

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

About This Manual

This manual tells how to perform day-to-day administration of your GemStone/S 64 Bit repository.

Installation instructions are included with your *GemStone/S 64 Bit Installation Guide*, which should be kept with this manual.

This manual is organized in three parts: initial configuration, day-to-day administration, and appendixes:

Part 1: System Configuration

- Chapter 1, “Configuring the GemStone Server,” tells how to adapt the GemStone central repository server to the needs of your application. Three sample configuration files are provided as starting points.
- Chapter 2, “Configuring Gem Session Processes,” tells how to configure the GemStone processes that provide the services to individual application clients.
- Chapter 3, “Connecting Distributed Systems,” explains the additional steps necessary to run GemStone in a networked environment. It includes examples of how to set up common configurations.

Part 2: System Administration

- Chapter 4, “Running GemStone,” tells how to start and stop the GemStone system, how to troubleshoot startup problems, how to deal with unexpected shutdowns, and how to bulk-load objects.
- Chapter 5, “User Accounts and Security,” introduces the tools available for administration tasks and details how to log in to the repository, and how to create, modify, and remove GemStone user accounts. It also tells how to configure GemStone login security by restricting valid passwords, imposing password and account age limits, and monitoring intrusion attempts.
- Chapter 6, “Managing Repository Space,” gives procedures for managing the repository itself: checking free space, adding space, and controlling its growth. It also how to recover from disk-full conditions.
- Chapter 7, “Managing Transaction Logs,” gives procedures for setting up the optional full incremental logging, managing log space, and archiving the log files.
- Chapter 8, “Monitoring GemStone,” explains where the system logs are located, how to audit the repository, and how to monitor the performance of the GemStone server and its clients using GemStone Smalltalk methods.
- Chapter 9, “Making and Restoring Backups,” gives procedures for making a GemStone full backup while the repository is in use, and for using backups and transaction logs to restore the repository.
- Chapter 10, “Managing Growth,” presents the main concepts underlying garbage collection in GemStone and tells when, how, and why to invoke the garbage collection mechanisms.

Part 3: Appendixes

- Appendix A, “GemStone Configuration Options,” explains how GemStone uses configuration files and describes each configuration option.
- Appendix B, “GemStone Utility Commands,” describes each of the GemStone-supplied commands defined for use by the GemStone system administrator.
- Appendix C, “Network Resource String Syntax,” lists the syntax for network resource strings, which allow you to specify the host machine for a GemStone file or process.
- Appendix D, “GemStone Kernel Objects,” lists the GemStone-supplied objects that are present in your repository after the GemStone system has been successfully installed.

- Appendix D, “GemStone Kernel Objects,” lists the contents of the GemStone system and data directories that are created when the GemStone software is installed.
- Appendix E, “Environment Variables,” lists all environment variables used by GemStone, including those that are reserved.
- Appendix F, “statmonitor and VSD Reference,” describes how to use the performance-tuning tools statmonitor and VSD.

Terminology Conventions

This document uses the following terminology:

- The term “GemStone” is used to refer both to the product, GemStone/S 64 Bit, or previous GemStone/S server products; and to the company, GemStone Systems, Inc.

Typographical Conventions

This document uses the following typographical conventions:

- Operating system and Topaz commands are shown in **bold** typeface. For example:

copydbf

- Smalltalk methods, GemStone environment variables, operating system file names and paths, listings, and prompts are shown in `monospace` typeface. For example:

`markForCollection`

- Interactive dialogue from GemStone is shown in an underlined monospace typeface. For example:

successful login

- Lines you type are distinguished from system output by boldface type:

`topaz> set gemstone myStone`

- Place holders that are meant to be replaced with real values are shown in *italic* typeface. For example:

StoneName.conf

In formal syntax listings, these additional conventions are used:

- Literals are shown in **bold** typeface. For example:

tcp

- Optional arguments and terms are enclosed in square brackets. For example:

[dbfName]

- Braces { } mean 0 or more modifiers. For example:

{modifier}

In this example you may list as many modifiers as you wish, but they are not required.

- Alternative arguments and terms are separated by a vertical bar (pipe). For example:

gemStoneName | netLdiName

In this example you must specify one name, but not both.

Technical Support

GemStone provides several sources for product information and support. The product-specific manuals and online help provide extensive documentation, and should always be your first source of information. GemStone Technical Support engineers will refer you to these documents when applicable.

GemStone Web Site: <http://support.gemstone.com>

GemStone's Technical Support website provides a variety of resources to help you use GemStone products. Use of this site requires an account, but registration is free of charge. To get an account, just complete the Registration Form, found in the same location. You'll be able to access the site as soon as you submit the web form.

The following types of information are provided at this web site:

Help Request allows designated support contacts to submit new requests for technical assistance and to review or update previous requests.

Documentation for GemStone/S 64 Bit is provided in PDF format. This is the same documentation that is included with your GemStone/S 64 Bit product.

Release Notes and **Install Guides** for your product software are provided in PDF format in the Documentation section.

Downloads and **Patches** provide code fixes and enhancements that have been developed after product release. Most code fixes and enhancements listed on the GemStone Web site are available for direct downloading.

Bugnotes, in the Learning Center section, identify performance issues or error conditions that you may encounter when using a GemStone product. A bugnote describes the cause of the condition, and, when possible, provides an alternative means of accomplishing the task. In addition, bugnotes identify whether or not a fix is available, either by upgrading to another version of the product, or by applying a patch. Bugnotes are updated regularly.

TechTips, also in the Learning Center section, provide information and instructions for topics that usually relate to more effective or efficient use of GemStone products. Some Tips may contain code that can be downloaded for use at your site.

Community Links provide customer forums for discussion of GemStone product issues.

Technical information on the GemStone Web site is reviewed and updated regularly. We recommend that you check this site on a regular basis to obtain the latest technical information for GemStone products. We also welcome suggestions and ideas for improving and expanding our site to better serve you.

You may need to contact Technical Support directly for the following reasons:

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

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

When contacting GemStone Technical Support, please be prepared to provide the following information:

- Your name, company name, and GemStone/S license number
- The GemStone product and version you are using

- The hardware platform and operating system you are using
- A description of the problem or request
- Exact error message(s) received, if any

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

For non-emergency requests, the support website is the preferred way to contact Technical Support. Only designated support contacts may submit help requests via the support website. If you are a designated support contact for your company, or the designated contacts have changed, please contact us to update the appropriate user accounts.

Email: support@gemstone.com

Telephone: (800) 243-4772 or (503) 533-3503

Requests for technical assistance may also be submitted by email or by telephone. We recommend you use telephone contact only for more serious requests that require immediate evaluation, such as a production system that is non-operational. In these cases, please also submit your request via the web or email, including pertinent details such error messages and relevant log files.

If you are reporting an emergency by telephone, select the option to transfer your call to the technical support administrator, who will take down your customer information and immediately contact an engineer.

Non-emergency requests received by telephone will be placed in the normal support queue for evaluation and response.

24x7 Emergency Technical Support

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

Training and Consulting

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

- Customized consulting services can help you make the best use of GemStone products in your business environment.

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

**Part I:
Configuring
GemStone/S 64 Bit**

Chapter 1. Configuring the GemStone Server

1.1 Configuration Overview	1-3
The Server Configuration File	1-4
Sample Configurations	1-5
Recommendations About Disk Usage	1-9
Why Use Multiple Drives?	1-9
When to Use Raw Partitions	1-10
Developing a Failover Strategy	1-11
1.2 How to Establish Your Configuration	1-11
Gathering Application Information	1-12
Planning Operating System Resources	1-12
Estimating Memory Needs	1-12
Estimating Swap Space Needs	1-13

Estimating File Descriptor Needs	1-13
Reviewing Kernel Tunable Parameters	1-14
Checking the System Clock	1-14
To Set the Page Cache Options and the Number of Sessions	1-15
Shared Page Cache	1-15
Stone's Private Page Cache.	1-17
Procedure	1-17
Diagnostics.	1-18
To Configure the Repository Extents	1-19
Estimating Extent Size	1-19
Choosing the Extent Location	1-20
Setting a Maximum Size for an Extent	1-20
Pregrowing Extents to a Fixed Size	1-21
Allocating Data to Multiple Extents.	1-23
To Configure the Transaction Logs	1-27
Choosing a Logging Mode	1-27
Estimating the Log Size.	1-28
Choosing the Log Location and Size Limit.	1-29
To Configure Server Response to Gem Fatal Errors.	1-30
To Set File Permissions for the Server	1-30
Recommended: Use the Setuid Bit	1-31
Alternative: Use Group Write Permission	1-33
Access to Other Server Files	1-33
1.3 How to Set Up a Raw Partition	1-34
Sample Raw Partition Setup.	1-35
Changing Between Files and Raw Partitions	1-36
Moving an Extent to a Raw Partition	1-36
Moving an Extent to the File System	1-36
Moving Transaction Logging to a Raw Partition	1-37
Moving Transaction Logging to the File System.	1-37
1.4 How to Access the Server Configuration at Run Time	1-38
To Access Current Settings at Run Time	1-38
To Change Settings at Run Time	1-39
1.5 How to Tune Server Performance	1-41
To Tune the Shared Page Cache.	1-41
Adjusting the Cache Size	1-41
Matching Spin Lock Limit to Number of Processors	1-41
Clustering Objects That Are Accessed Together.	1-42

- To Reduce Excessive Swapping 1-42
- To Control Checkpoint Frequency 1-42
- Adding Page Servers 1-43
 - To Add AIO Page Servers 1-44
 - Do You Need Free Frame Page Servers? 1-44
 - To Add Free Frame Page Servers 1-45
 - Process Free Frame Caches 1-45
- 1.6 How to Run a Second Repository 1-46
- 1.7 How to Operate a Duplicate Server 1-48

Chapter 2. Configuring Gem Session Processes

- 2.1 Overview 2-1
 - Linked and RPC Applications 2-2
 - The Session Configuration File 2-3
- 2.2 How to Configure Gem Session Processes 2-4
 - Gathering Application Information 2-4
 - Planning Operating System Resources 2-4
 - Estimating Memory Needs 2-4
 - Estimating Swap Space Needs 2-5
 - Estimating File Descriptor Needs 2-5
 - Reviewing Kernel Tunable Parameters 2-6
 - Set the Gem Configuration Options. 2-6
 - To Set Ownership and Permissions for Session Processes 2-6
 - To Set Access for Linked Applications 2-7
 - To Set Access for All Other Applications 2-8
 - To Set Access to Other Files 2-8
- 2.3 How to Access the Configuration at Run Time 2-9
 - To Access Current Settings at Run Time 2-9
 - To Change Settings at Run Time. 2-9
- 2.4 How to Tune Session Performance 2-11
 - To Tune the Temporary Object Space. 2-11
 - To Tune the Private Page Cache 2-12
 - To Reduce Excessive Swapping of Sleeping Sessions 2-12
- 2.5 How to Install a Custom Gem 2-12

Chapter 3. Connecting Distributed Systems

3.1 Overview	3-2
GemStone NetLDIs	3-4
NetLDI Names.	3-4
GemStone Page Servers	3-5
GemStone Network Objects	3-5
Shared Page Cache in Distributed Systems	3-6
Disrupted Communications	3-7
3.2 How to Arrange Network Security.	3-9
Default: Password Authentication	3-12
Using a .netrc File	3-12
Using the Application Interface	3-13
Using an NRS #auth modifier	3-13
Alternative: Guest Mode With a Captive Account	3-14
3.3 How to Use Network Resource Strings	3-15
To Set a Default NRS	3-15
To Use copydbf Between Nodes	3-16
3.4 How to Set Up a Remote Session	3-18
To Duplicate the GemStone Installation	3-18
To Share a GemStone Directory	3-19
Configuration Examples	3-19
To Run a Linked Application on a Remote Node	3-20
To Run the Gem Session Process on the Stone's Node	3-23
To Run the Gem and Stone on Different Nodes	3-25
To Run the Application, Gem, and Stone on Three Nodes	3-28
Troubleshooting Remote Logins	3-30
If You Still Have Trouble	3-31
Check NetLDI Log Files	3-32

Part II: Administering GemStone/S 64 Bit

Chapter 4. Running GemStone

4.1 How to Start the GemStone Server	4-2
--	-----

To Start GemStone	4-2
To Troubleshoot Stone Startup Failures	4-3
Missing or Invalid Key File	4-4
Shared Page Cache Cannot Be Attached	4-4
Extent Missing or Access Denied	4-5
Extent Open by Another Process.	4-5
Extent Already Exists	4-6
Other Extent Failures	4-6
Transaction Log Missing	4-7
Repository Failure	4-7
Other Startup Failures.	4-8
4.2 How to Start a NetLDI	4-8
To Troubleshoot NetLDI Startup Failures	4-9
4.3 To List Running Servers	4-10
4.4 How to Start a GemStone Session.	4-10
To Define a GemStone Session Environment	4-11
To Start a Linked Session.	4-11
To Start an RPC Session	4-13
To Troubleshoot Session Login Failures	4-14
4.5 How to Identify Sessions Logged In	4-16
4.6 How to Shut Down the Object Server and NetLDI	4-17
4.7 How to Recover from an Unexpected Shutdown	4-19
Normal Shutdown Message	4-20
Disk Failure or File System Corruption.	4-20
Shared Page Cache Error	4-21
Fatal Error Detected by a Gem.	4-21
Some Other Shutdown Message.	4-22
No Shutdown Message	4-22
4.8 How to Bulk-Load Objects.	4-22
4.9 Considerations for Large Repositories	4-23
Disk Space and Commit Record Backlogs	4-24
Handling signals indicating a commit record backlog	4-25

Chapter 5. User Accounts and Security

5.1 The Administrative Accounts	5-2
5.2 Defining Your GemStone Environment	5-3

5.3 User Accounts	5-3
UserProfiles	5-3
Predefined Users	5-7
The SystemUser Account	5-8
The UserProfile and Session Symbol Lists	5-8
The UserGlobals SymbolDictionary	5-9
The Globals SymbolDictionary	5-9
The Published SymbolDictionary	5-9
Sharing Objects	5-10
5.4 Using GemBuilder for Administration	5-11
Logging in Through GemBuilder	5-11
Finding the GemBuilder Administration Tools	5-14
Committing Your Changes	5-14
Logging Out	5-14
Administering User Accounts	5-16
To List Existing Users	5-17
To Add a User	5-17
To Remove a User	5-19
To Change a Password	5-20
To Change a User's Privileges	5-20
To Add a Dictionary to a Symbol List	5-21
To Examine a User's Group Memberships	5-23
To Add a User to a Group	5-23
To Remove a User from a Group	5-23
Administering Segment Authorizations	5-23
To Find Out Who Is Authorized to Read or Write in a Segment	5-23
To Change the Authorization of a Segment	5-25
To Change a User's Default Segment	5-26
5.5 Using Topaz for Administration	5-27
Logging in Through Topaz	5-27
The Printit Command	5-28
The Commit Command	5-29
Administering User Accounts	5-29
To List Existing Users	5-29
To Add a User	5-30
To Change Your Own Password	5-31
To Change Another User's Password	5-32

- To Examine a User's Privileges. 5-32
- To Assign a Privilege to a User. 5-33
- To Revoke a User's Privilege 5-33
- To Redefine a User's Privileges 5-34
- To Add a SymbolDictionary to Your Own Symbol List. . . . 5-34
- To Add a SymbolDictionary to Someone Else's Symbol List . 5-35
- To Remove a SymbolDictionary from Your Own Symbol List . . 5-35
- To Remove a SymbolDictionary from Someone Else's Symbol List. 5-36
- To Examine a User's Group Memberships 5-36
- To Add a User to a Group 5-36
- To Remove a User from a Group. 5-37
- To List All Members of a Group 5-37
- To Remove a User Group. 5-38
- To Modify Someone's User ID 5-38
- To Remove an Account 5-38
- Administering Segment Authorizations 5-39
 - To Find Out Who Is Authorized to Read or Write in a Segment . 5-39
 - To Change the Authorization of a Segment. 5-40
 - To Remove a Group from a Segment's Authorization List . . 5-41
 - To Change a User's Default Segment 5-41
- 5.6 How to Configure GemStone Login Security 5-42
 - To Constrain the Choice of Passwords 5-42
 - Disallowing Particular Passwords. 5-44
 - Disallowing Reuse of Passwords. 5-44
 - To Require Periodic Password Changes 5-45
 - Providing Warning of Password Expiration 5-46
 - Finding Accounts With Password About to Expire 5-46
 - Finding Out When a Password Was Changed 5-46
 - To Disable Inactive Accounts 5-47
 - Finding Out When an Account Last Logged In 5-47
 - To Limit Logins Until Password Is Changed. 5-48
 - To Limit Concurrent Sessions by a Particular UserId 5-49
 - To Record Login Failures. 5-49
 - Disabling Further Login Attempts. 5-49
 - To Find Out Which Accounts Have Been Disabled 5-50
 - To Verify That an Account Is Disabled 5-51

To Find Out Why an Account Was Disabled	5-51
---	------

Chapter 6. Managing Repository Space

6.1 Repository Growth	6-2
6.2 How to Check Free Space	6-3
6.3 How to Add Extents	6-5
To Add an Extent While the Stone is Running	6-5
Possible Effects on Other Sessions.	6-5
Repository>>createExtent:	6-6
Repository>>createExtent: withMaxSize:	6-7
6.4 How to Remove an Extent.	6-7
6.5 How To Reallocate Existing Objects Among Extents.	6-8
To Reallocate Objects Among a Different Number of Extents	6-8
To Reallocate Objects Among the Same Number of Extents	6-9
6.6 How to Shrink the Repository.	6-10
6.7 How to Check Page Fragmentation	6-13
6.8 How to Recover from Disk-Full Conditions.	6-14
Repository Full	6-14

Chapter 7. Managing Transaction Logs

7.1 Overview	7-1
Logging Modes	7-3
Recovering from an Unexpected Shutdown	7-5
Restoring Transactions to a Backup.	7-5
How the Logs Are Used	7-6
7.2 How to Manage Full Logging	7-7
To Archive Logs.	7-8
To Add a Log at Run Time.	7-10
To Force a New Transaction Log	7-11
To Start Checkpoints	7-12
To Change to Partial Logging	7-12
7.3 How to Manage Partial Logging	7-13
To Change to Full Logging.	7-13
7.4 How to Recover from Tranlog-Full Conditions.	7-13
Transaction Log Space Full	7-13

Chapter 8. Monitoring GemStone

8.1 GemStone System Logs	8-2
GemStone Server Logs	8-2
Stone Log	8-3
Admin GcGem Logs.	8-4
Shared Page Cache Monitor Log.	8-4
Free Frame Page Server Log	8-5
AIO Page Server Log	8-5
Page Manager Log.	8-6
Reclaim GcGem Logs	8-6
Symbol Gem Log	8-7
Logs Related to Gem Sessions	8-7
NetLDI Logs	8-9
8.2 How to Audit the Repository	8-9
To Perform a Page Audit	8-10
To Perform an Object Audit and Repair	8-12
Object Audits Without Reclaim	8-13
Quick Audits	8-13
Performing the Object Audit	8-14
Error Recovery	8-15
Understanding Object Audit Statistics	8-16
8.3 Monitoring Performance	8-19
To Monitor Page Reads and Writes by a Session	8-19
To Monitor Cache Statistics	8-19
Cache Statistics.	8-23

Chapter 9. Making and Restoring Backups

9.1 Overview	9-2
9.2 How to Make an Online Extent Backup	9-3
9.3 How to Restore from an Online Extent Backup	9-7
9.4 How to Make a Smalltalk Full Backup	9-8
Additional Performance Tips	9-10
Backups and Garbage Collection.	9-11
To Create a Backup on a Remote Node	9-11
To Create a Backup in Multiple Files	9-12
To Create Compressed Backups	9-13

To Verify a Backup is Readable	9-14
To Examine the Backup Log	9-14
9.5 How to Restore from a Smalltalk Full Backup	9-14
Phase 1: Restore to the Point of the Backup	9-16
Phase 2: Restore Subsequent Transactions	9-19
Other Considerations.	9-22
Performance Tips	9-22
To Restore Multiple-File Backups	9-23
To Restore Backups from Tape	9-24
To Restore Logs to a Point in Time	9-25
Errors While Restoring Transaction Logs.	9-26
Precautions When Restoring a Subset of Transaction Logs.	9-30
9.6 How to Make an Offline Extent Backup	9-33
9.7 How to Restore from an Offline Extent Backup.	9-33
9.8 How to Recover After Repair of the File System	9-37
To Recover After a File System Repair With fsck	9-37
To Recover When a File System Must Be Restored	9-38
9.9 How To Manage a Warm Standby System	9-41

Chapter 10. Managing Growth

10.1 Basic Concepts.	10-2
Shadow or Dead?	10-3
What Happens to Garbage?	10-6
Admin and Reclaim GcGems	10-8
Different Ways to Collect Garbage	10-9
Where.	10-9
How.	10-9
GemStone's Garbage Collection Mechanisms	10-9
Both Marking and Reclaiming	10-9
Marking Only	10-10
Reclaiming Only.	10-10
10.2 Temporary Object Memory Garbage Collection	10-11
Examining and Allocating Temporary Memory Usage.	10-12
Signal on low memory condition	10-17
10.3 Epoch Garbage Collection	10-17
Determining the Epoch Length	10-20

- 10.4 markForCollection. 10-26
 - Reducing Impact on Other Sessions 10-28
 - Scheduling markForCollection. 10-28
- 10.5 Running the Admin GcGem 10-29
 - Stopping the Admin GcGem. 10-30
- 10.6 The FDC/MGC Process. 10-31
 - Fast FDC 10-33
 - Run markGcCandidatesFromFile:. 10-34
- 10.7 Reclaiming Pages 10-35
 - Configuring and Starting the Reclaim GcGems 10-36
 - Stopping the Reclaim GcGems. 10-40
 - To Identify Sessions Holding Up Page Reclamation 10-41
 - To Tune Reclamation 10-41
 - To Remove References to Large Objects 10-44
 - To Identify Large Objects in the Repository 10-44
 - To Search for References to an Object 10-45
 - To Remove References to an Object 10-46

Appendixes

Appendix A. GemStone Configuration Options

- How GemStone Uses Configuration Files A-2
 - Search for a System-Wide Configuration File A-2
 - Search for an Executable Configuration File A-5
 - Creating or Using a System Configuration File A-6
 - Creating an Executable Configuration File. A-6
 - Naming Executable Configuration Files A-7
 - Naming Conventions for Configuration Options A-9
- Configuration File Syntax A-9
 - Errors in Configuration Files. A-11
 - Syntax Errors. A-11
 - Option Value Errors. A-11
- Configuration Options A-11
 - DBF_ALLOCATION_MODE A-12
 - DBF_EXTENT_NAMES A-12

DBF_EXTENT_SIZES.	A-13
DBF_PRE_GROW.	A-13
DBF_SCRATCH_DIR.	A-14
DUMP_OPTIONS.	A-14
GEM_FREE_FRAME_CACHE_SIZE.	A-14
GEM_FREE_FRAME_LIMIT	A-14
GEM_GCI_LOG_ENABLED	A-15
GEM_HALT_ON_ERROR	A-15
GEM_IO_LIMIT.	A-15
GEM_KEEP_MIN_SOFTREFS.	A-16
GEM_MAX_SMALLTALK_STACK_DEPTH	A-16
GEM_PGSVR_FREE_FRAME_CACHE_SIZE	A-16
GEM_PGSVR_FREE_FRAME_LIMIT	A-17
GEM_PGSVR_UPDATE_CACHE_ON_READ	A-17
GEM_PRIVATE_PAGE_CACHE_KB.	A-17
GEM_RPCGCI_TIMEOUT.	A-18
GEM_SEND_STN_MSGS_VIA_PGSVR	A-18
GEM_SOFTREF_CLEANUP_PERCENT_MEM	A-18
GEM_TEMPOBJ_AGGRESSIVE_STUBBING	A-19
GEM_TEMPOBJ_CACHE_SIZE.	A-19
GEM_TEMPOBJ_INITIAL_SIZE	A-20
GEM_TEMPOBJ_MESPACE_SIZE	A-20
GEM_TEMPOBJ_OOPMAP_SIZE	A-20
GEM_TEMPOBJ_POMGEN_SIZE	A-21
KEYFILE	A-21
LOG_WARNINGS	A-21
SHR_NUM_FREE_FRAME_SERVERS.	A-22
SHR_PAGE_CACHE_LOCKED.	A-22
SHR_PAGE_CACHE_NUM_SHARED_COUNTERS	A-22
SHR_PAGE_CACHE_NUM_PROCS.	A-22
SHR_PAGE_CACHE_SIZE_KB.	A-23
SHR_SPIN_LOCK_COUNT	A-23
SHR_TARGET_FREE_FRAME_COUNT.	A-24
STN_ADMIN_GC_SESSION_ENABLED	A-24
STN_ALLOCATE_HIGH_OOPS	A-24
STN_CHECKPOINT_INTERVAL	A-25
STN_COMMIT_QUEUE_THRESHOLD	A-25
STN_COMMIT_TOKEN_TIMEOUT	A-25
STN_COMMITS_ASYNC	A-26

STN_CR_BACKLOG_THRESHOLD	A-26
STN_DISABLE_LOGIN_FAILURE_LIMIT	
STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT.	A-26
STN_DISKFULL_TERMINATION_INTERVAL.	A-27
STN_EPOCH_GC_ENABLED	A-27
STN_FREE_FRAME_CACHE_SIZE.	A-27
STN_FREE_SPACE_THRESHOLD	A-27
STN_GEM_ABORT_TIMEOUT	A-28
STN_GEM_LOSTOT_TIMEOUT	A-28
STN_GEM_TIMEOUT	A-29
STN_HALT_ON_FATAL_ERR	A-29
STN_LOG_LOGIN_FAILURE_LIMIT	
STN_LOG_LOGIN_FAILURE_TIME_LIMIT	A-30
STN_LOOP_NO_WORK_THRESHOLD.	A-30
STN_MAX_AIO_RATE.	A-31
STN_MAX_AIO_REQUESTS	A-31
STN_MAX_SESSIONS	A-32
STN_MAX_VOTING_SESSIONS	A-32
STN_NUM_GC_RECLAIM_SESSIONS	A-32
STN_NUM_LOCAL_AIO_SERVERS	A-33
STN_OBJ_LOCK_TIMEOUT.	A-33
STN_PAGE_REMOVAL_THRESHOLD	A-33
STN_PRIVATE_PAGE_CACHE_KB	A-34
STN_REMOTE_CACHE_TIMEOUT	A-34
STN_SHR_TARGET_PERCENT_DIRTY	A-34
STN_SIGNAL_ABORT_CR_BACKLOG	A-35
STN_TRAN_FULL_LOGGING	A-35
STN_TRAN_LOG_DEBUG_LEVEL.	A-36
STN_TRAN_LOG_DIRECTORIES	A-36
STN_TRAN_LOG_LIMIT	A-36
STN_TRAN_LOG_PREFIX.	A-36
STN_TRAN_LOG_SIZES.	A-37
STN_TRAN_Q_TO_RUN_Q_THRESHOLD.	A-37
Miscellaneous Internal Parameters	A-37
#LogOriginTime.	A-38
#SessionInBackup	A-38
#StnCurrentTranLogDirId	A-38
#StnCurrentTranLogNames	A-38
#StnLogGemErrors	A-38

#StnLoginsSuspended	A-38
#StnMaxReposSize	A-38
#StnMaxSessions	A-39
#StnSunsetDate	A-39
#StnTranLogOriginTime	A-39

Appendix B. GemStone Utility Commands

copydbf	B-2
gslis	B-7
pageaudit	B-9
removedbf	B-10
startcachewarmer	B-11
startnetldi	B-13
startstone	B-15
stopnetldi	B-17
stopstone	B-18
topaz	B-19
waitstone	B-20

Appendix C. Network Resource String Syntax

Overview	C-1
Defaults	C-2
Notation	C-3
Syntax	C-4

Appendix D. GemStone Kernel Objects

Users	D-1
Dictionaries	D-2
Non-Numeric Constants	D-3
Numeric Constants	D-3
Repository and Segments	D-3
Global Collections	D-5

Current TimeZone	D-7
Zoneinfo	D-8
Utilities	D-9

Appendix E. Environment Variables

Public Environment Variables.	E-1
System Variables Used by GemStone	E-5
Reserved Environment Variables	E-5

Appendix F. statmonitor and VSD Reference

Using statmonitor and VSD	F-1
Starting VSD and statmonitor	F-2
Loading an existing statmonitor output file	F-3
Viewing VSD online help	F-5
Maintaining a current view of the data file	F-5
Starting statmonitor from VSD and viewing current data	F-5
Viewing Statistics	F-7
Customizing Your Chart	F-11
Filter.	F-13
Using VSD Chart Templates	F-13
Statmonitor command line syntax	F-14
VSD Files.	F-16
.vsdrc	F-16
.vsdconfig	F-16
.vsdtemplates	F-16
statFilter examples.	F-17

Index

***Part I:
Configuring
GemStone/S 64 Bit***

—
|

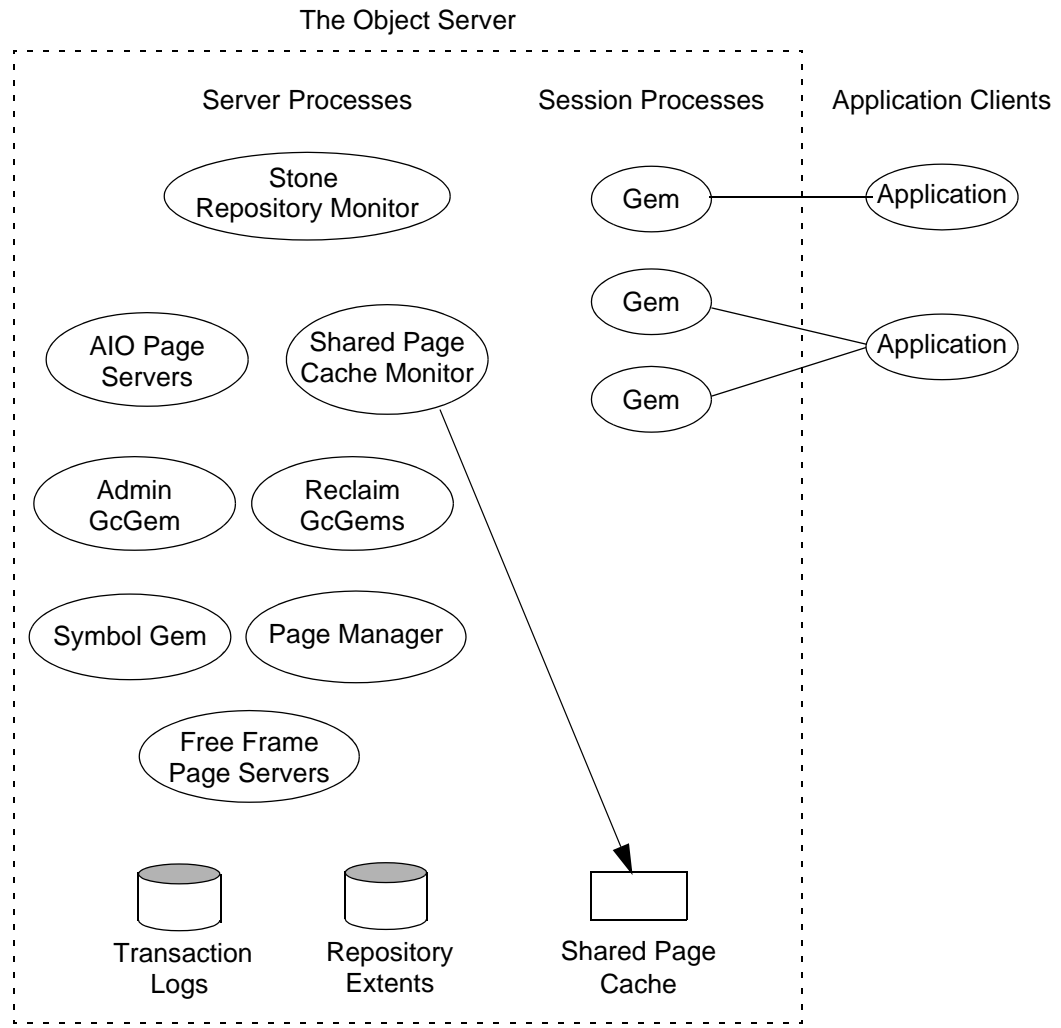
Configuring the GemStone Server

Figure 1.1 shows the basic GemStone/S 64 Bit architecture as seen by its administrator. The object server can be thought of as having two active 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 to application clients.

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. For information about configuring session processes for clients, refer to Chapter 2.

The elements shown in Figure 1.1 can be distributed across multiple nodes to meet your application's needs. For information about establishing distributed servers, refer to Chapter 3.

Figure 1.1 The GemStone Object Server



1.1 Configuration Overview

Figure 1.1 shows the key parts that define the server configuration:

- The *Stone repository monitor* process acts as a resource coordinator. It synchronizes critical repository activities and ensures repository consistency.
- The *shared page cache monitor* creates and maintains a *shared page cache* for the GemStone server. The monitor balances page allocation among processes, ensuring that a few users or large objects do not monopolize the cache. The size of the shared page cache is configurable and should be scaled to match the size of the repository and the number of concurrent sessions.
- The *AIO page servers* perform asynchronous I/O for the Stone repository monitor. Their primary tasks are to update the extents periodically and to pre-allocate (grow) the extents at startup when that feature is enabled. The default configuration uses one AIO page server, but additional ones can be specified for systems having several extents.
- The *Admin GcGem* is a Gem server process that is dedicated to performing the administrative garbage collection tasks under supervision of the Stone. Each repository can have up to one Admin GcGem process running.
- The *Reclaim GcGems* perform page reclaim operations on both shadow objects and dead objects. On a running GemStone system, there may be between 0 and n Reclaim GcGems present, where n is the number of extents in the repository.
- The *Symbol Gem* is a Gem server background process that is responsible for creating all new Symbols, based on session requests that are managed by the Stone.
- The *Page Manager* is a background process that assists the Stone with page disposal in coordination with the remote page caches.
- The *Free Frame Page Servers* are Gem server processes that are dedicated to the task of adding free frames to the free frame list, from which a Gem can take as needed. The default configuration uses one free frame page server, but you can configure as many as 30 free frame page server processes.
- Objects are stored on the disk in one or more *extents*, which can be files in the file system, data in raw partitions, or a mixture. The location of each extent is configurable.
- *Transaction logs* permit recovery of committed data if a system crash occurs. They also reduce disk activity by eliminating the need to flush to the extents all data pages written by each transaction. The optional *full logging mode* allows

transaction logs to be used with GemStone backups for full recovery of committed transactions in the event of media failure.

The transaction logs should reside on a different disk drive (spindle) from the extents, and neither should be on a drive that contains the operating system swap space (sometimes called page space).

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,” tells how to specify an alternate configuration file and how to use supplementary files to adjust the system-wide configuration for a specific GemStone executable. The appendix also describes each of the configuration options.

Here is a brief summary of important facts about the configuration file:

- Lines that begin with `#` are comments. Settings supplied as comments are the same as the default values. You can easily change the configuration by altering the option value and moving the `#` symbol to the line previously in force.
- Options that begin with “`GEM_`” are read only by Gem session processes at the time they start. Chapter 2, “Configuring Gem Session Processes,” describes their use.
- Options that begin with “`SHR_`” are read both by the Stone repository monitor and by the first Gem session process to start on a node remote from the Stone. These options configure the local shared page cache.
- Most other options (those not beginning with “`GEM_`” or “`SHR_`”) are read by the Stone repository monitor. If another GemStone process needs that information, it is exchanged through a TCP/IP connection with the Stone.
- If an option is defined more than once, only the last definition is used. Certain run-time configuration changes, such as the addition of an extent, cause the repository monitor to append new configuration statements to the file. Be sure to check the end of a configuration file for possible entries that override earlier ones.

Sample Configurations

This section describes three sample configurations that you can use as a starting point. Although the configurations differ in a number of ways, the primary difference is in the size of application they accommodate.

All three sample configurations are derived from the initial configuration file that is installed, `$GEMSTONE/data/system.conf`. The initial configuration provides a convenient way to begin evaluation or development with a minimum of system resources.

NOTE

The sample configuration files contain only the modifications that define a particular sample configuration. These modifications override the default settings. For a complete list of options, see `$GEMSTONE/data/system.conf`.

- | | |
|--------|--|
| Small | <p>Handles I/O more efficiently than the initial configuration by using separate drives (spindles) for the extents and transaction logs. A larger page cache supports more users. Full transaction logging provides real-time incremental backup.</p> <p>Edit the sample configuration file <code>\$GEMSTONE/examples/admin/small.conf</code> to specify the filename of your extent and the directory names for transaction logs.</p> |
| Medium | <p>Uses raw disk partitions for possibly increased throughput. It accommodates more users and a larger repository.</p> <p>Edit the sample configuration file <code>\$GEMSTONE/examples/admin/medium.conf</code> to show the raw partition name and size for your extent, and the partition names and sizes for transaction logs.</p> |
| Large | <p>Uses multiple extents to accommodate a repository of 100 GB.</p> <p>Edit the sample configuration file <code>\$GEMSTONE/examples/admin/large.conf</code> to show the raw partition names and sizes for your extents, and the partition names and sizes for transaction logs. Each extent should be on a separate spindle.</p> |

To choose a sample configuration, select a column in Table 1.1 by matching the characteristics of your application to those shown. The table shows the corresponding changes to be made to the default configuration file, `$GEMSTONE/data/system.conf`. (Sample changes have already been made to the three configuration files in `$GEMSTONE/examples/admin`.)

NOTE

The contents of the sample configuration files may vary from the information in Table 1.1.

Table 1.1 also gives recommended configurations for repository extents and transaction logs. Some of these, such as the use of raw disk partitions, depend on the size of the application.

NOTE

Large systems will almost certainly require additional tuning. Very large systems will probably need to be distributed across several powerful servers. If you lack the necessary expertise, consider consulting GemStone Professional Services.

If you want more information about any of these settings, see the detailed instructions for establishing your own configuration beginning on page 1-11.

Very Large Configurations

On UNIX machines, there is an upper limit to the number of processes that you can run before performance becomes unacceptable. For very large configurations, we recommend a separate Gem server machine with a remote shared page cache set up specifically to run Gem sessions, with a high bandwidth connection between the repository server and the Gem server.

Table 1.1 Settings for Selected Configurations

Characteristic or Configuration Option	Server Configuration		
	Small	Medium	Large
Application Characteristics			
Maximum number of user sessions	12	250	1500
Repository size	100 MB	10 GB	100 GB
System Requirements			
Typical number of CPUs	1-2	2-4	8+
RAM	512 MB	8000 MB	128000 MB
Kernel shared memory	26 MB	1600 MB	64500 MB
Number of disk drives	3	13	35
Configuration Settings			
STN_MAX_SESSIONS	40	280	1658
SHR_PAGE_CACHE_SIZE_KB	75000	1500000	64000000
STN_PRIVATE_PAGE_CACHE_KB	(default)	16384	32768
STN_CHECKPOINT_INTERVAL	300	600	900
STN_CR_BACKLOG_THRESHOLD	(default)	500	3000
STN_SIGNAL_ABORT_CR_BACKLOG	20	400	2800
STN_FREE_SPACE_THRESHOLD	10	100	100
STN_NUM_LOCAL_AIO_SERVERS	1	4	10
SHR_NUM_FREE_FRAME_SERVERS	1	2	5

Table 1.1 Settings for Selected Configurations (Continued)

Characteristic or Configuration Option	Server Configuration		
	Small	Medium	Large
Approximate Memory Usage			
Stone repository monitor (MB)	20	40	100
Each Gem session process ^a (MB)	20	20	25
Extents			
DBF_EXTENT_NAMES	(1 file)	(8 files)	(25 files)
DBF_EXTENT_SIZES	(Unlimited)	(1995 each) ^b	(3995 each) ^b
DBF_PRE_GROW	False	True	True
DBF_ALLOCATION_MODE	(Not used)	10,10,10,...	10,10,10,...
Transaction Logs			
STN_TRAN_FULL_LOGGING	True	True	True
STN_TRAN_LOG_DIRECTORIES	(2 directories)	(5 raw partitions)	(10 raw partitions)
STN_TRAN_LOG_SIZES	50,50	499 ^b each	1995 ^b each
Stone Response to Gem Fatal Errors			
STN_HALT_ON_FATAL_ERROR	False ^c	False ^c	False ^c
Garbage Collection			
STN_NUM_GC_RECLAIM_SESSIONS	1	2	5
STN_ADMIN_GC_SESSION_ENABLED	True	True	True
^a Depends on the value of GEM_TEMPOBJ_CACHE_SIZE (default=10 MB). ^b For best performance, set DBF_EXTENT_NAMES and STN_TRAN_LOG_SIZES to slightly less than the actual size of the partition. The values given for extents are based on 2 GB partitions in the Medium configuration and 4 GB partitions in the Large configuration. ^c For development and testing, a setting of True is recommended. For deployed systems, a setting of False is recommended.			

Recommendations About Disk Usage

You can enhance server performance by distributing the repository files on multiple disk drives. Under certain circumstances and for certain operating systems, placing the data in raw disk partitions rather than in a file system can enhance performance.

Why Use Multiple Drives?

Efficient access to GemStone repository files requires that the server node have at least three disk drives (that is, three separate spindles or physical volumes) to reduce I/O contention. For instance:

- One spindle for swap space and UNIX (GemStone executables can also reside here).
- One spindle for the repository extent, perhaps with a lightly accessed file system sharing the drive.
- One spindle for transaction logs (with least two raw partitions or directories) and possibly user file systems if they are only lightly used for non-GemStone purposes.

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

1. Keep extents and transaction logs separate from operating system swap space. Don't place either extents or logs on any spindle that contains a swap partition; doing so drastically reduces performance.
2. Place the transaction logs on a spindle that does not contain extents. Placing logs on a different spindle from extents increases the transaction rate for updates while reducing the impact of updates on query performance. It's OK to place multiple logs on the same spindle because only one log file is active at a time.

NOTE

Under operating systems that use volume managers, you need to be aware of how logical volume groups are assigned to disk drives (physical volumes). You should try to assign each of the above (swap, extents, and transaction logs) to a different disk drive.

3. To benefit from multiple extents on multiple spindles, you must use weighted allocation mode. If you use sequential allocation, multiple extents provide no benefit. For details about weighted allocation, see “Allocating Data to Multiple Extents” on page 1-23.
4. In addition, if you decide to use more than one AIO page server, you’ll need to keep extents on several different spindles. You’ll derive no advantage from multiple page servers unless they can write different pages to different extents simultaneously, instead of contending for the same disk drive head.

When to Use Raw Partitions

Each raw partition (sometimes called a raw device or raw logical device) is like a single large sequential file, with one extent or one transaction log per partition. The use of raw disk partitions can yield better performance, depending on how they are used and the balancing of system resources:

- Placing transaction logs on raw disk partitions almost certainly yields better performance.
- Placing extents on raw disk partitions can yield better performance to the degree that doing so reduces swapping. However, if sufficient RAM is available for file system buffers and the shared page cache, better performance may be obtained by placing the extents in the file system.

The use of raw partitions for transaction logs is essential for achieving the highest transaction rates in an update-intensive application because such applications primarily are writing sequentially to the active transaction log. Using raw partitions can as much as double the maximum achievable rate by avoiding the extra file system operations necessary to ensure that each log entry is recorded on the disk.

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. If your application has a high transaction volume, you are likely to find that increasing the number log partitions makes the task of archiving the logs easier.

For information about using raw partitions, see “How to Set Up a Raw Partition” on page 1-34.

Developing a Failover Strategy

In choosing a failover strategy, consider the following needs:

- Applications that cannot tolerate the loss of committed transactions should mirror the transaction logs (using OS-level tools) and use full transaction logging. A mirrored transaction log on another device allows GemStone to recover from a read failure when it starts up after an unexpected shutdown. The optional full logging mode allows transactions to be rolled forward from a GemStone full backup to recover from the loss of an extent.
- Applications that require rapid recovery from the loss of an extent (that is, without the delay of restoring from a backup) should replicate all extents on other devices, preferably through hardware means, in addition to mirroring transaction logs. Restoring a large repository (many GB) from a backup may take hours.

Hardware replication may provide the best solution if the following points are kept in mind while designing the system:

- 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. Use of RAID devices (redundant array of inexpensive drives) or striped file systems that cannot efficiently support both random and sequential access may reduce overall performance. Simple disk mirroring may give better results.
- Transaction logs use sequential access exclusively, so the devices can be optimized for that access.
- Avoid volume managers that combine space on multiple physical drives. For GemStone, such configurations may result in *less* efficient access to the repository. The use of raw devices is preferred for transaction logs.

1.2 How to Establish Your Configuration

Configuring the GemStone object server involves the following steps:

1. Gather application specifics about the size of the repository and the number of sessions that will be logged in simultaneously.
2. Plan the operating system resources that will be needed: memory and swap (page) space.

3. Set the size of the GemStone shared page cache and the number of sessions to be supported.
4. Configure the repository extents.
5. Configure the transaction logs.
6. Set GemStone file permissions to allow necessary access while providing adequate security.

Gathering Application Information

When you begin configuring GemStone, be sure to have the following information at hand:

- The number of simultaneous sessions that will be logged in to the repository (in some applications, each user can have more than one session logged in).
- The approximate size of your repository. It's also helpful, but not essential, to know the approximate number of objects in the repository.

This information is central to the sizing decisions that you must make.

Planning Operating System Resources

GemStone needs adequate memory and swap space to run efficiently. It also needs adequate kernel resources—for instance, kernel parameters can limit the size of the shared page cache or the number of sessions that can connect to it.

Estimating Memory Needs

The amount of memory required to run your GemStone server depends mostly on the size of the repository and the number of users who will be logged in to active GemStone sessions at one time. These needs are in addition to the memory required for UNIX and other software.

- The Stone and related processes need between 45 and 325 MB for the configurations shown in Table 1.1 (on page 1-7). That amount of memory is only for the server processes.
- The shared page cache should be increased in proportion to the overall size of your repository. Typically it should be approximately 10% of the repository size to provide adequate performance. In Table 1.1, the size ranges from 10 MB to 10 GB.

On a node that is dedicated to running GemStone, we recommend in general that you allocate approximately one-third to one-half of your total system

RAM to the shared page cache. If it is not a dedicated node, you may need to reduce the size to avoid excessive swapping.

- Each Gem session process needs at least 30 MB of memory on the node where it runs. (See the discussion of memory needs for session processes on page 2-4.) Each Gem process that runs on a remote (client) node also needs about 0.25 MB on the server node for a GemStone page server process that accesses the repository extents.

Estimating Swap Space Needs

To provide reasonable flexibility, the total swap space on your system (sometimes called page space) in general should be at least twice the system RAM. For example, a system with 256 MB of RAM should have at least 512 MB of swap space. The command to find out how much swap space is available (**swap**, **swapinfo**, **pstat**, or **lsvg**) depends on your operating system. Your *GemStone/S 64 Bit Installation Guide* contains an example for your platform.

Swap space should not be on a disk drive that contains any of the GemStone repository extent files. In particular, do not use operating system utilities like **swap** or **swapon** to place part of the swap space on a disk that also contains the GemStone extents or transaction logs.

If you want to determine the additional swap space needed just for GemStone, use the memory requirements derived in the preceding section, including space for the number of sessions you expect. These figures will approximate GemStone's needs beyond the swap requirement for UNIX and other software such as the X Window System.

Estimating File Descriptor Needs

When they start, most GemStone processes attempt to raise their file descriptor limit from the default (soft) limit to the hard limit set by the operating system. In the case of the Stone repository monitor, the processes that raise the limit this way are the Stone itself and two of its child processes, the AIO page server and the Admin GcGem. The Stone uses file descriptors this way:

- 9 for stdin, stdout, stderr, and internal communication
- 2 for each user session that logs in
- 1 for each local extent or transaction log within a file system
- 2 for each extent or transaction log that is a raw partition
- 1 for each extent or transaction log that is on a remote node

You can cause the above processes to set a limit less than the system hard limit by setting the `GEMSTONE_MAX_FD` environment variable to a positive integer. A value of 0 disables attempts to change the default limit.

The shared page cache monitor always attempts to raise its file descriptor limit to equal its maximum number of clients plus five for `stdin`, `stdout`, `stderr`, and internal communication. The maximum number of clients is set by the `SHR_PAGE_CACHE_NUM_PROCS` configuration option (page A-22).

Reviewing Kernel Tunable Parameters

UNIX kernel parameters limit the interprocess communication resources that GemStone can obtain. It's helpful to know what the existing limits are so that you can either stay within them or plan to raise the kernel limits. There are four parameters of primary interest:

- The maximum size of a shared memory segment (typically `shmmax` or a similar name) limits the size of the shared page cache for each repository monitor.

For information about platform-specific limitations on the size of the shared page cache, refer to Chapter 1 of your *GemStone/S 64 Bit Installation Guide*.

- The maximum number of semaphores per semaphore id limits the number of sessions that can connect to the shared page cache, because each session uses one semaphore. (Typically this parameter is `semmsl` or a similar name, although it is not tunable under all operating systems.)
- The maximum number of users allowed on the system (typically `maxusers` or a similar name) can limit the number of logins and sometimes also is used as a variable in the allocation of other kernel resources by formula. In the latter case, you may need to set it somewhat larger than the actual number of users.
- The hard limit set for the number of file descriptors can limit the total number of logins and repository extents, as described previously.

How you determine the existing limits depends on your operating system. If the information is not readily available, proceed anyway. A later step shows how to verify that the shared memory and semaphore limits are adequate for the GemStone configuration you chose.

Checking the System Clock

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. The time comparisons use GMT. It is not necessary to adjust GemStone for changes to and from daylight savings time in the United States.

To Set the Page Cache Options and the Number of Sessions

Configure the shared page cache and the Stone's private page cache according to the size of the repository and the number of sessions that will connect to it simultaneously. Then use a GemStone utility to verify that the UNIX kernel will support this configuration.

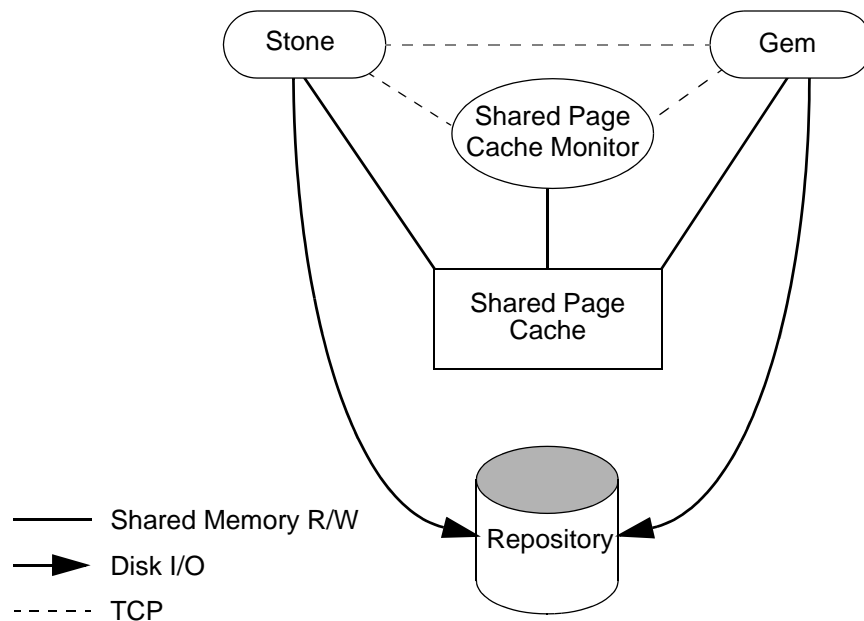
Shared Page Cache

The GemStone shared page cache system consists of two components, the shared page cache itself and a monitor process (`shrpcmonitor`). Figure 1.2 shows the connections between these two and the main GemStone components when GemStone runs on a single node. There is no direct connection between the shared page cache and the repository.

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 repository monitor 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 name of the shared page cache monitor ordinarily is derived from the name of the Stone repository monitor and the hostid in "dot" format; for instance, `gs64stone@192.83.233.25`. Some utilities, such as `gslist`, translate the address to the node's name before displaying it.

Each shared page cache is associated with exactly one Stone process and repository, and a Stone may never have more than one shared page cache on the same node. The Stone spawns the shared page cache automatically during startup. If other Gem session processes on the same node need to access that repository, they must connect to the same shared page cache and monitor process to ensure cache coherency. Use of the shared page cache reduces disk I/O and improves performance.

Figure 1.2 Shared Page Cache Configuration

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, thereby reducing disk I/O, while not inducing excessive swapping at the operating system level. Three factors are important in estimating the size (the minimum cache size in all cases should be 20 MB):

1. The number of objects in the repository.

For best performance, the entire object table should be in shared memory. At a minimum, we recommend that you allow room for one-third to one-half of the object table in the cache. Each object uses five bytes in the table.

2. The size of the repository. At a minimum, we recommend that you keep at least 25% of the repository in the cache. Although this factor is application-dependent, increasing the amount in the cache will improve performance.

3. The number of users and the size of their transactions. Again, this factor is specific to your application, depending (among other things) on how frequently users will be committing their transactions.

The cache size thus estimated is only a starting point for system configuration. It uses a percentage of the repository to estimate the size of the working set of objects, which can vary drastically depending on the application's design. In addition, the degree to which your application clusters objects that are likely to be accessed together can have a significant impact on space used in the cache.

Once your application is running, you can tune the cache size by monitoring the free space. See "To Monitor Page Reads and Writes by a Session" on page 8-19, especially the statistic `NumberOfFreeFrames`.

NOTE

For information about platform-specific limitations on the size of the shared page cache, see Chapter 1 of your GemStone/S 64 Bit Installation Guide.

Stone's Private Page Cache

As the Stone repository monitor allocates resources to each session, it stores the information in its private page cache. The size of this cache is set by the `STN_PRIVATE_PAGE_CACHE_KB` configuration option (page A-34). The default size of 2 MB is sufficient in almost all circumstances. If you think you might need to adjust this setting, please contact GemStone Technical Support.

Procedure

To configure the shared page cache, follow these steps:

- Step 1.** Set the `SHR_PAGE_CACHE_SIZE_KB` configuration option (page A-23), using Table 1.1 (on page 1-7) or your own estimate derived above (remember to convert to KB). For instance, for the Medium configuration's 1.5 GB cache:

```
SHR_PAGE_CACHE_SIZE_KB = 1500000;
```

- Step 2.** If the number of sessions will be greater than 40, increase the `STN_MAX_SESSIONS` configuration option (page A-32) accordingly.

Make sure the `SHR_PAGE_CACHE_NUM_PROCS` option (page A-22) is set to its default (-1), which causes GemStone to calculate a value based on `STN_MAX_SESSIONS`.

For instance:

```
STN_MAX_SESSIONS = 50;
SHR_PAGE_CACHE_NUM_PROCS = -1;
```

Step 3. Use GemStone's **shmem** utility to verify that your UNIX 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 either the `SHR_PAGE_CACHE_NUM_PROCS` setting or (if that option is set to -1) the greater of 15 or (number of extents + 3).

For instance, for the values used in Steps 1 and 2:

```
% touch /tmp/shmem
% $GEMSTONE/install/shmem /tmp/shmem 1500000 55
% 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 of about 20 bytes per KB of cache space plus about 20 KB per session in `SHR_PAGE_CACHE_NUM_PROCS`. The actual shared memory segment in this example would be 104865792 bytes (your system might produce slightly different results).

Diagnostics

The shared page cache monitor creates or appends to a log file, *gemStoneNamePid*`pcmon.log`, in the same directory as the log for the Stone repository monitor. The *Pid* portion of the name is the monitor's process id. In case of difficulty, check for this log (the cache monitor removes the log if the cache shuts down normally).

The operating system kernel must be configured appropriately on each node running a shared page cache. If **startstone** or a remote login fails because the shared cache cannot be attached, check *gemStoneName.log* and *gemStoneNamePid*`pcmon.log` for detailed information.

The following configuration settings are checked at startup:

- The kernel shared memory resources must be enabled and sufficient to provide the page space specified by `SHR_PAGE_CACHE_SIZE_KB` plus the cache overhead.

The kernel semaphore resources must also be sufficient to provide an array of size `SHR_PAGE_CACHE_NUM_PROCS + 1` semaphores.

Use the **shmem** utility to test the settings (see Step 3 on page 1-18). If multiple Stones are being run concurrently on the same node, each Stone requires a separate set of semaphores and separate semaphore id.

- Sufficient file descriptors must be available at startup to provide one descriptor for each of the `SHR_PAGE_CACHE_NUM_PROCS` processes plus an overhead of five. Compare your `SHR_PAGE_CACHE_NUM_PROCS` configuration setting to the operating system file descriptor limit per process.

On operating systems that permit it, the shared page cache monitor attempts to raise the descriptor soft limit to the number required. In some cases, raising the limit may require superuser action to raise the hard limit or to reconfigure the kernel.

To Configure 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 0.5 MB of free space for each session that will be logged in simultaneously—if necessary, the extent will be expanded to provide this much headroom.

Example 1.1 Extent Size Including Working Space

Size of repository	=	1 GB
Free-Space Allowance .5 MB * 20 sessions	=	10 MB
Fragmentation Allowance 1 GB * 15%	=	150 MB
Total with Working Space	=	1.16 GB

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. For information, see “How to Recover from Disk-Full Conditions” on page 6-14. (The default setting for `STN_FREE_SPACE_THRESHOLD` is 1 MB; see page A-27.)

For planning purposes, you should allow additional disk space for making GemStone backups (if you do not use tape) and for duplicating the repository when upgrading to a new release. A GemStone backup typically occupies 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. The following example uses two raw disk partitions (your partition names will be different):

```
DBF_EXTENT_NAMES = /dev/rdisk/c1t3d0s5, /dev/rdisk/c2t2d0s6;
```

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 (page A-13). 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, or until 32 terabytes (32,000 GB) have been allocated.

NOTE

For best performance using raw partitions, the maximum size should be slightly 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 1995 MB.

Each size entry is for the corresponding entry in DBF_EXTENT_NAMES (page A-12). Use a comma to mark the position of an extent for which you do not want to specify a limit. For example, the following settings are for two extents of 500 MB each in raw partitions.

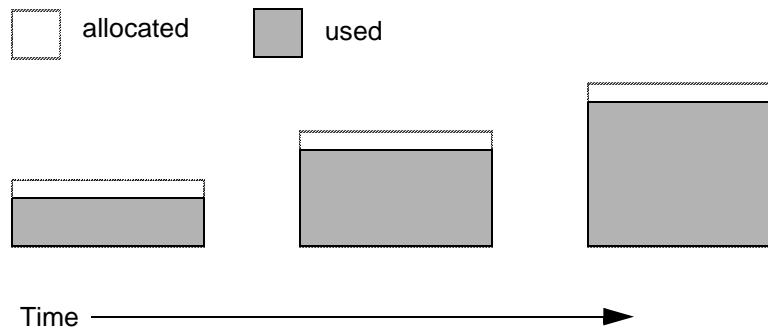
```
DBF_EXTENT_NAMES = /dev/rdisk/c1t3d0s5, /dev/rdisk/c2t2d0s6;
DBF_EXTENT_SIZES = 498, 498;
```

The maximum size of an extent is limited by the operating system and platform, but under no circumstances can be larger than 16 GB. For specific information about system dependencies, see the comment in the configuration file for the parameter DBF_EXTENT_SIZES.

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 1.3), your application must incur this overhead and then initialize the added extent pages.

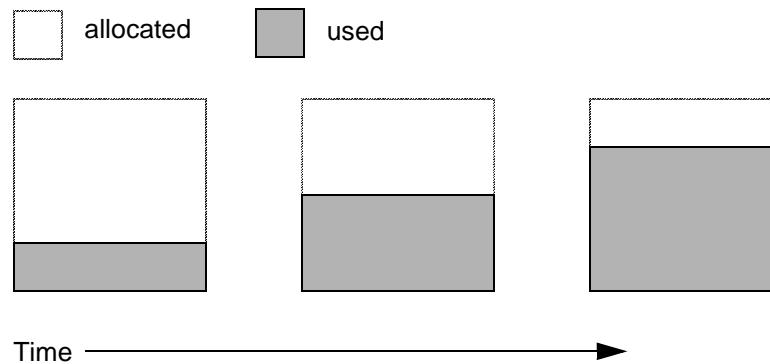
Figure 1.3 Growing an Extent on Demand



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. When `DBF_PRE_GROW` is set to `True`, the Stone repository monitor obtains the necessary space when it creates a new extent or starts with an extent that is smaller than the specified size.

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. It is especially useful in performance benchmarking. Pregrowing extents also provides a simple way to reserve space on a disk for a GemStone extent.

Figure 1.4 Pregrowing an Extent



The disadvantages of pregrowing extents are that it takes longer to start GemStone the next time or to add an extent dynamically, and unused disk space allocated to pregrown extents is unavailable for other purposes.

Two configuration options work together to pregrow extents:

- `DBF_PRE_GROW` (page A-13) enables the operation. When `DBF_PRE_GROW` is set to `True`, the Stone repository monitor obtains the necessary space when it creates a new extent or starts with an extent that is smaller than the specified size.
- `DBF_EXTENT_SIZES` (page A-13) sets the size limit individually for each extent. For optimal performance, the size should be slightly smaller than the actual size of the disk partition.

To pregrow extents, set both of these configuration options (and remove the comment symbol from DBF_PRE_GROW line). For example:

```
DBF_EXTENT_SIZES = 498, 498;  
DBF_PRE_GROW = TRUE;
```

Allocating Data to Multiple Extents

If your application is query-intensive, you should consider dividing the repository into multiple extents and placing each extent on a separate spindle so that accesses can overlap. When GemStone schedules disk writes, it assumes that you have done so. Because several extents can be active at once, putting them on the same spindle limits the maximum update rate and causes updating transactions to have unexpected impact on the response time for queries.

The DBF_ALLOCATION_MODE configuration option (page A-12) 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. If you have placed each extent on a separate disk drive as recommended, the 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 1.5 on page 1-24.) 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 1.6 (on page 1-25).

Figure 1.5 Sequential Allocation

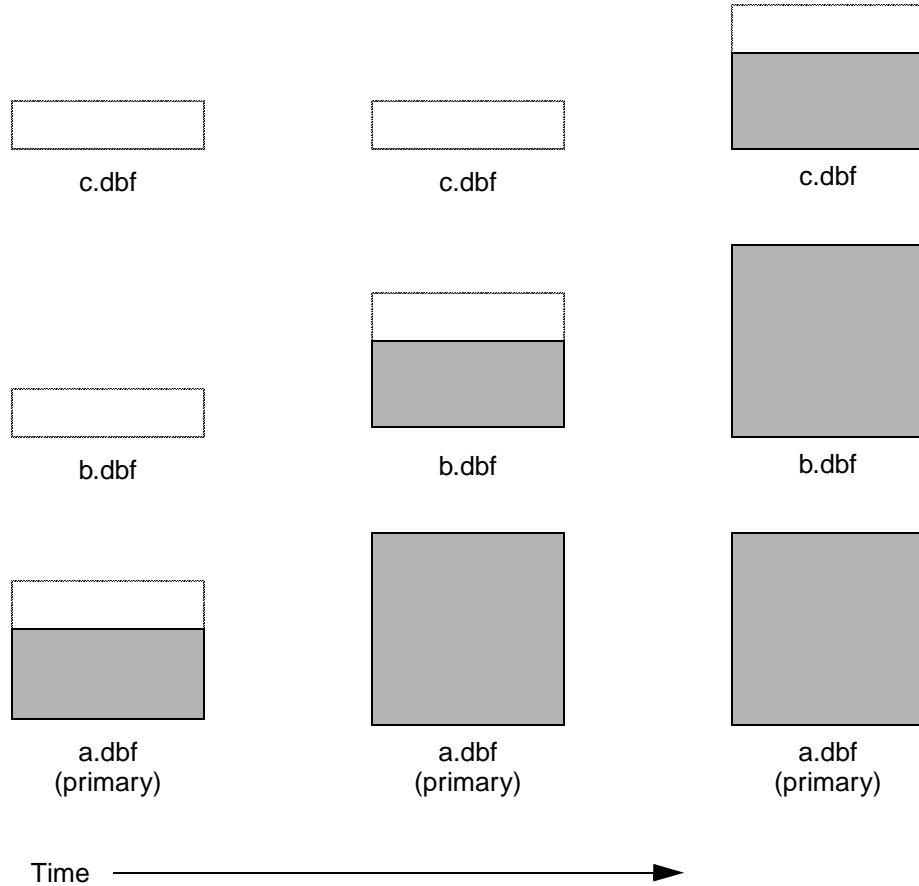
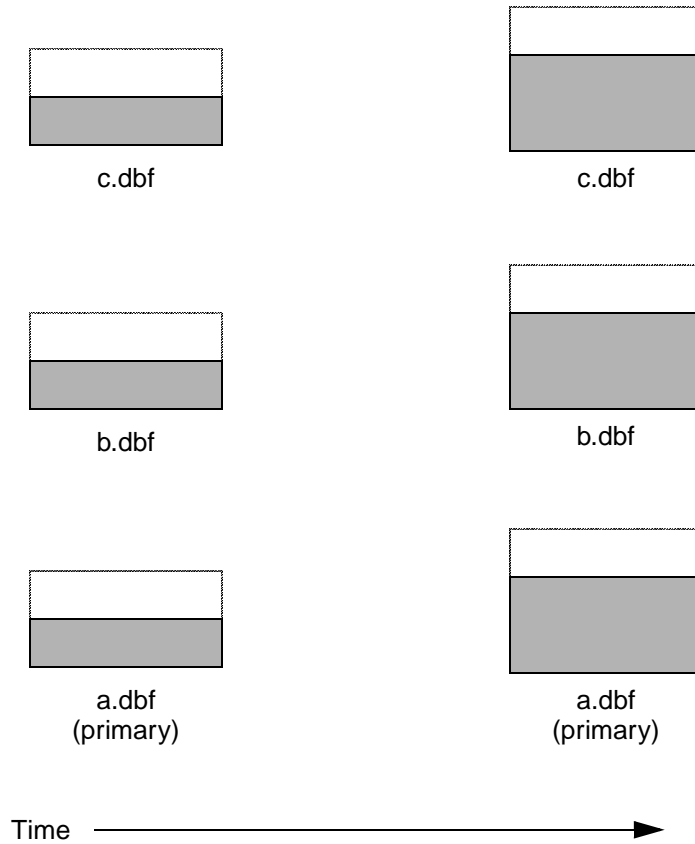


Figure 1.6 Equally Weighted Allocation

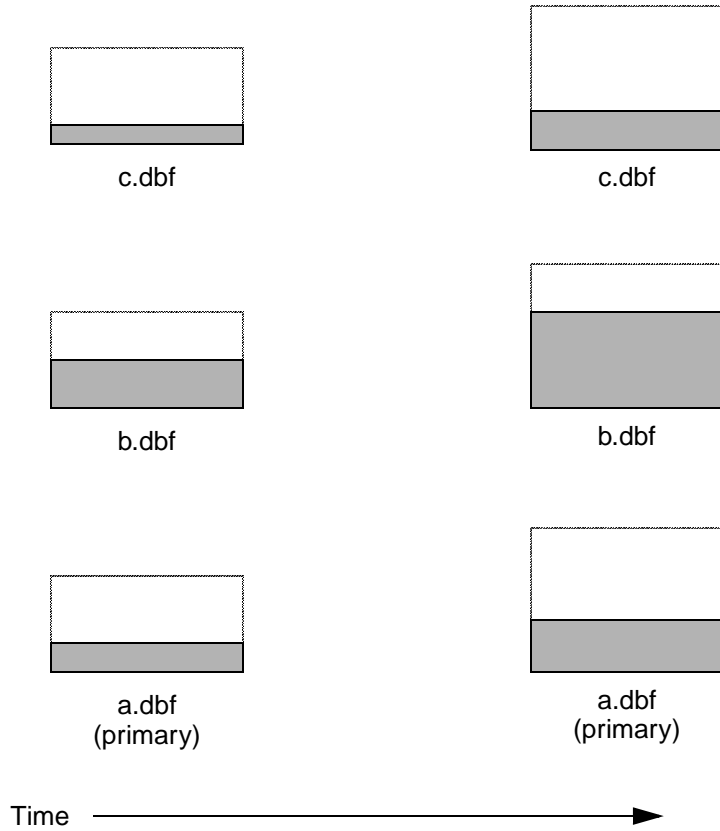
`DBF_ALLOCATION_MODE = 10,10,10;`



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 1.7 shows the result.

Figure 1.7 Proportionally Weighted Allocation

`DBF_ALLOCATION_MODE = 12,20,8;`



You can modify the relative extent weights by editing your GemStone configuration file and modifying the values listed for `DBF_ALLOCATION_MODE`. You can also change `DBF_ALLOCATION_MODE` to `SEQUENTIAL` without harming the system. The new values you specify take effect the next time you start the GemStone system.

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 *GemStone/S 64 Bit 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).

To Configure the Transaction Logs

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

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

Choosing a Logging Mode

GemStone provides two modes of transaction logging:

- *Full logging* 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 be used in restoring the repository from a GemStone backup.
- *Partial logging* is the default mode, and is intended for use during evaluation or early stages of application development. Partial logging is also recommended during bulk loading of the repository. 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.

To enable full transaction logging, change the configuration setting to True and restart the Stone repository monitor:

```
STN_TRAN_FULL_LOGGING = TRUE;
```

CAUTION

The only backups to which you can apply transaction logs are those made while the repository is in full logging mode. If you change to full logging, be sure to make a GemStone backup as soon as circumstances permit.

Changing the logging mode from full to partial logging requires special steps. See “To Change to Partial Logging” on page 7-12.

For general information about the logging mode and the administrative differences, see “Logging Modes” on page 7-3.

Estimating the Log Size

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 you have configured GemStone for full transaction logging (that is, `STN_TRAN_FULL_LOGGING` is set to True), you must allow sufficient space to log all transactions until you next archive the logs.

CAUTION

If the Stone exhausts the 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 1.2 provides an estimate for an application that expects to generate 4500 transactions a day.

At any point, the method `Repository>>oldestLogFileIdForRecovery` identifies the oldest log file needed for recovery from the most recent checkpoint, if the Stone were to crash. Log files older than the most recent checkpoint (the default maximum interval is 5 minutes) are needed only if it becomes necessary to restore the repository from a backup. Although the older logs can be retrieved from archives, you may want to keep them online until the next GemStone full backup, if you have sufficient disk space.

Example 1.2 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 (the default), 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. (In partial logging mode, transaction logs are used only after an unexpected shutdown to recover transactions since the last checkpoint.) If you use the default configuration, you should provide space for at least two logs of 2 MB each.

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 spindle than operating system swap space.
- Where possible, keep the extents and transaction logs on separate spindles—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) can yield somewhat 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.

GemStone requires at least two log locations (directories or raw partitions) so it can switch to another when the current one is filled. When you set the log locations in the configuration file, you should also check their size limit.

Although the default size of 10 MB is adequate in some situations, update-intensive applications should consider a larger size (at least 25 to 50 MB) 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 50 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, 50;
```

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

To Configure 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. By default, the Stone halts and dumps a core image if it receives notification from a Gem that the Gem process died with a fatal error that would cause the Gem to dump core. By stopping both the Gem and the Stone at this point, the possibility of repository corruption is minimized. During application development, it may be helpful to know exactly what the Stone was doing when the Gem went down.

For deployed production systems, we recommend that you change the default in the Stone's configuration file so that the Stone will attempt to keep running:

```
STN_HALT_ON_FATAL_ERROR = FALSE;
```

To Set File Permissions for the Server

The primary consideration in setting file permissions for the Server is to protect the repository extents. All reads and writes should be done through GemStone

repository executables: the Stone and its child processes (the shared page cache monitor, AIO page server, Admin and Reclaim GcGems, Symbol Gem, Page Manager, and Free Frame Page Server) and the Gem session processes. Chapter 3 describes the use of additional page servers to read and write extents in networked systems.

Recommended: Use the Setuid Bit

The tightest repository security is obtained by having the extents and the repository executables owned by a single UNIX account, using the UNIX setuid bit (S bit) on the executable files, and making the extents writable only by that account. The S bit causes a process to belong to the owner you specify for the file.

Table 1.2 shows the recommended file settings, where *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 repository monitor (Stone) must be logged in as *gsadmin* or have execute permission. The Stone and shared page cache will belong to the owner you specify for the files.

Table 1.2 Recommended Resource and Process Permissions for the Server

Resource or Process ^a	Protection Mode	File Owner	File Group	Process Runs As	Comments
repository extents	-rw-----	gsadmin	gsgroup		Users read and write repository through GemStone processes. The Stone sets up the page cache in shared memory.
shared memory	-rw-rw----	gsadmin	gsgroup		
stoned	-r-sr-xr-x	gsadmin	gsgroup	gsadmin	AIO page server and GC Gem are spawned by stoned and can access repository as the <i>gsadmin</i> account.
pgsvrmain	-r-sr-xr-x	gsadmin	gsgroup	gsadmin	
gem	-r-sr-xr-x	gsadmin	gsgroup	gsadmin	

^a Ownership and permissions for the netluid 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 1.2. To set them manually, do the following as root:

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

The protection mode for the shared memory segment is fixed in GemStone.

You must take similar steps to provide access for repository clients, which are presented in Chapter 2. See “To Set Ownership and Permissions for Session Processes” on page 2-6.

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 *gs*group is a group of your choice:

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

Sites that run the linked version of GemBuilder 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.

Access to Other Server Files

GemStone creates log files and other special files in several locations, which are described below. 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` All users should have read/write/execute access to the directories `/opt/gemstone/log` and `/opt/gemstone/locks` on each node. (On some systems, these directories may be located in `/usr/gemstone`.)

By default, NetLDIs (Network Long Distance Information processes) create log files in the `log` directory.

GemStone processes that have a name for each instance (currently the Stone, shared page cache monitor, and NetLDI) create lock files in the `locks` directory.

`system.conf` The user who owns the Stone process must have write permission for the Stone configuration file, which by default is `$GEMSTONE/data/system.conf`. If certain configuration changes are made while the Stone is running, the Stone updates that file. For instance, the Stone must record run-time changes such as those made by `Repository>>createExtent:` so that it can restart later in the correct configuration.

1.3 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.

The instructions in this section are incomplete intentionally. You will need to work with your system administrator to locate a partition of suitable size for your extent or transaction log. Consult the system documentation for guidance as necessary.

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/r1vol1`. Most operating systems have a utility or administrative interface that can assist you in identifying existing partitions; some examples are **prtvtoc**, **dkinfo**, and **vgdisplay**. 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. If necessary, unmount a file system that is currently mounted and edit the system table. 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. For example, set the permissions to 600, and set the owner to the GemStone administrator.

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 Raw Partition Setup

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.)

```
% 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 1 to 2 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 A-12).

Set `DBF_EXTENT_SIZES` for this extent to be 1 to 2 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 A-36). 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.

1.4 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 category Configuration File Access let you examine the system-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 2-9.)

Here is a partial example of the Stone configuration report:

```
topaz 1> printit
System stoneConfigurationReport asReportString
%
#SHR_SPIN_LOCK_COUNT      1200
#StnDisableLoginFailureTimeLimit      15
#StnDisableLoginFailureLimit      15
#SHR_PAGE_CACHE_LOCKED  false
...
```

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

To Change Settings at Run Time

The class method `System class>>configurationAt: aName put: aValue` in category `Runtime Configuration Access` lets you change the value of the internal run-time parameters in Table 1.3 if you have the appropriate privileges. The parameters that can be changed are those for which `ConfigurationParameterDict at: aName` returns a negative `SmallInteger`. All changeable parameters require that `aValue` be a `SmallInteger`.

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. For additional guidance about run-time changes to specific parameters, see Appendix A, "GemStone Configuration Options."

Table 1.3 Server Configuration Parameters Changeable at Run Time

Configuration File Option	Internal Parameter
SHR_SPIN_LOCK_COUNT	#SpinLockCount ^a
STN_ADMIN_GC_SESSION_ENABLED	#StnAdminGcSessionEnabled ^b
STN_CHECKPOINT_INTERVAL	#StnCheckpointInterval ^a
STN_COMMIT_QUEUE_THRESHOLD	#StnCommitQueueThreshold ^a
STN_CR_BACKLOG_THRESHOLD	#StnCrBacklogThreshold ^a
STN_DISABLE_LOGIN_FAILURE_LIMIT	#StnDisableLoginFailureLimit ^c
STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT	#StnDisableLoginFailureTimeLimit ^c
STN_DISKFULL_TERMINATION_INTERVAL	#StnDiskFullTerminationInterval ^a
STN_EPOCH_GC_ENABLED	#StnEpochGcEnabled ^b
STN_FREE_SPACE_THRESHOLD	#StnFreeSpaceThreshold
STN_GEM_ABORT_TIMEOUT	#StnGemAbortTimeout ^a
STN_GEM_LOSTOT_TIMEOUT	#StnGemLostOtTimeout ^a
STN_GEM_TIMEOUT	#StnGemTimeout ^a
STN_HALT_ON_FATAL_ERR	#StnHaltOnFatalErr ^a
STN_LOG_LOGIN_FAILURE_LIMIT	#StnLogLoginFailureLimit ^c
STN_LOG_LOGIN_FAILURE_TIME_LIMIT	#StnLogLoginFailureTimeLimit ^c
STN_LOOP_NO_WORK_THRESHOLD	#StnLoopNoWorkThreshold ^a

Table 1.3 Server Configuration Parameters Changeable at Run Time (Continued)

Configuration File Option	Internal Parameter
STN_MAX_AIO_RATE	#StnMntMaxAioRate ^a
STN_MAX_VOTING_SESSIONS	#StnMaxVotingSessions ^a
STN_NUM_GC_RECLAIM_SESSIONS	#StnNumGcReclaimSessions ^b
STN_OBJ_LOCK_TIMEOUT	#StnObjLockTimeout
STN_REMOTE_CACHE_TIMEOUT	#StnRemoteCacheTimeout ^a
STN_PAGE_REMOVAL_THRESHOLD	#StnPageRemovalThreshold
STN_SHR_TARGET_PERCENT_DIRTY	#ShrPcTargetPercentDirty ^a
STN_SIGNAL_ABORT_CR_BACKLOG	#StnSignalAbortCrBacklog ^b
STN_TRAN_LOG_LIMIT	#StnTranLogLimit ^a
STN_TRAN_Q_TO_RUN_Q_THRESHOLD	#StnTranQToRunQThreshold ^a
(none)	#StnLoginsSuspended ^d

^a Can be changed only by SystemUser.

^b Requires GarbageCollection privilege.

^c Requires OtherPassword privilege.

^d Requires SystemControl privilege.

The following example first obtains the value of #GcSessionEnabled. The parameter is one that can be changed at run time by SystemUser:

```
topaz 1> printit
ConfigurationParameterDict at: #AdminGcSessionEnabled
%
-9
topaz 1> printit
System configurationAt: #AdminGcSessionEnabled put: 0
%
0
```

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

1.5 How to Tune 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 the shared page cache, reduce swapping, and control disk activity caused by repository checkpoints.

To Tune 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

Adjust the `SHR_PAGE_CACHE_SIZE_KB` configuration option (page A-23) according to the total number of objects in the repository and the number accessed at one time. For best performance, the entire object table should be in shared memory. At a minimum, we recommend that the shared page cache should be large enough to hold one-third to one-half of the object table and all the pages on which currently used objects reside.

In general, the more of your repository you can hold in your cache, the better your performance will be. The size of the cache should not exceed one-half of your physical memory.

You should review the configuration recommendations given earlier (“Estimating the Size of the Shared Page Cache” on page 1-16) in light of your application’s design and usage patterns. Estimates of the number of objects queried or updated are particularly useful in tuning the cache.

You can use the shared page cache statistics for a running application to monitor the amount of unused space. See “To Monitor Page Reads and Writes by a Session” on page 8-19, especially the statistic `FreeFrameCount`.

Matching Spin Lock Limit to Number of Processors

The `SHR_SPIN_LOCK_COUNT` configuration option (page A-23) 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 you leave `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 *GemStone/S 64 Bit Programming Guide*.

To Reduce Excessive Swapping

Be careful not to make the shared page cache so large that it forces excessive swapping. If your node is dedicated to running GemStone, our general recommendation (as given on page 1-16) is that you use up to one-half of its RAM for the cache. If it is not a dedicated node, you may need to limit the cache size to something smaller than one-half of the RAM. When the node's memory is also used for non-GemStone processes, your shared page cache may need to be swapped, which would affect performance adversely.

Excessive swapping also can be caused by the need to awaken (and swap in) sleeping sessions that are outside of a transaction. When the Stone repository monitor reaches its configured limit, it sends a `SignaledAbort` message to the sleeping session, causing it to be swapped back into memory. Each such session must awaken long enough to update its view of the repository.

You can reduce this type of swapping activity by increasing the `STN_SIGNAL_ABORT_CR_BACKLOG` configuration option (page A-35). For example, you might determine a desired interval between `SignaledAbort` messages, and then use your application's commit rate to calculate the setting of `STN_SIGNAL_ABORT_CR_BACKLOG`. You may need to take the following additional steps:

- Increase the `STN_PRIVATE_PAGE_CACHE_KB` configuration option to $(\text{STN_SIGNAL_ABORT_CR_BACKLOG} + 10) / 30$ MB.
- Increase the size of the shared page cache. The default backlog of 20 commit records requires about 1 MB, assuming that a typical small transaction occupies about 50 KB.

If your configuration uses multiple extents in the file system, you may be able to reduce swapping by limiting the size of file system buffers. Some operating systems do not support this restriction.

To Control Checkpoint Frequency

Each checkpoint guarantees that the committed state of the repository has been written to the extent files. If the checkpoints interfere 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 (see page A-25). 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, so you may want to increase the size of these logs to reduce the frequency of checkpoints.
- In partial logging mode, checkpoints also are triggered by any transaction that is larger than `STN_TRAN_LOG_LIMIT`, which sets the size of the largest entry that is to be appended to the transaction log (see page A-36). The default limit is 100 KB 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.

For information about tuning `STN_TRAN_LOG_LIMIT` in partial logging mode, see the discussion of the `CheckpointCount` cache statistic on page 8-25.

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.

Suspending Checkpoints

You can call the method `System class>>suspendCheckpointsForMinutes:` to suspend checkpoints for a given number of minutes, or until `System class>>resumeCheckpoints` is executed. (To execute these Smalltalk methods, you must have the required GemStone privilege, as described in Chapter 5, “User Accounts and Security”.)

Adding Page Servers

GemStone uses page servers for three purposes:

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

Page servers referred to as *AIO page servers* perform all three functions. By default, at least one such page server is running at all times, though you can add more as

needed. In addition, you can add one or more *free frame page servers*: page servers dedicated only to the third task in the list above, adding free frames to the free list.

Under certain circumstances, 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.

To Add AIO Page Servers

By default the Stone spawns a single page server process on its local node to perform asynchronous I/O (AIO) between the shared page cache and the extents. This page server ordinarily is the process that updates extents on the local node during a checkpoint. (In some cases, the Stone may use additional page servers temporarily during startup to pregrow multiple extents.)

If your configuration has multiple extents on separate disk spindles, and you are trying to achieve the maximum possible commit rate, consider increasing the number of AIO page servers in use during ordinary operation. You can do this by changing the `STN_NUM_LOCAL_AIO_SERVERS` configuration option (page A-33).

Additional page servers are unlikely to benefit you unless the host computer has at least two CPUs, and the disk drive hardware supports concurrent writes to multiple extents. For multiple page servers to be effective, they must be able to execute at the same time and write to disk at the same time.

Do You Need Free Frame Page Servers?

A Gem can get free frames either from the free list (the quick way), or, if sufficient free frames are not listed, by scanning the shared page cache for a free frame instead. (What constitutes sufficient free frames is determined by the configuration parameter `GEM_FREE_FRAME_LIMIT`; for details, see page A-14.)

If a Gem has to spend a large proportion of its time scanning the shared page cache, its performance may be unacceptable. Under these circumstances, extra free frame page servers can sometimes help. On a single-CPU system, one extra free frame page server might be all that's required; for systems with multiple CPUs, you may wish to start one at a time, checking statistics, until the problem is resolved.

By default, when you start the Stone, it tries to spawn one free frame page server process on its local node. Free frame page servers require a running NetLDI

process, however; if the NetLDI process is not already running on the node, the attempt fails and the Stone writes a message to its log file.

Certain cache statistics can help you determine whether additional free frame page servers will improve system performance. (For details about these and other statistics, see “Cache Statistics” on page 8-23.)

- If Gems have to scan the shared page cache for free frames, the cache statistic `FramesFromFindFree` will be greater than zero. If this is the case—especially if it significantly greater—consider starting one or more free frame page servers.
- If the `FreeFrameCount` is consistently lower than the `FreeFrameLimit`, a free frame page server might help (though other factors also enter into play).

If `FramesAddedToFreeList` rises significantly after starting a free frame page server, the new page server has indeed benefited you; likewise, if `FramesFromFindFree` is reduced to zero, or near zero.

To Add Free Frame Page Servers

You can change the number of free frame page server processes that will be started when the shared page cache is created by setting a configuration parameter, `SHR_NUM_FREE_FRAME_SERVERS`.

Default: 1

Minimum: 1

Maximum: (`SHR_PAGE_CACHE_NUM_PROCS` – 5)

As mentioned previously, the NetLDI process must already be running in order for the free frame page servers to be started properly.

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 process 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` and `GEM_PGSVR_FREE_FRAME_CACHE_SIZE`, respectively.

For each of these, a value of 0 disables the free frame cache (the Gem or remote page server process acquires frames one at a time). A value of -1 means use the default value: 0 (for shared page caches less than 100 MB) or 10 (for shared page caches of 100 MB or greater).

Default: -1
Minimum: -1
Maximum: 63

1.6 How to Run 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 also start its own shared page cache monitor and a set of other processes, as described on page 1-3.)
- 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 serves all Stones and Gem session processes on a given node.
- Multiple Stones can share a single installation directory, provided that you create separate repository extents, transaction logs, and configuration files. If performance is a concern, the first step should be to isolate each Stone's data directory and the system swap space on separate drives. Then, review the discussion "Recommendations About Disk Usage" on page 1-9.

The following example shows the steps necessary to create a separate repository for application development (identified here by the prefix `dev`). This repository will run in parallel with the initial repository that you installed by following the instructions in the *GemStone/S 64 Bit Installation Guide*.

In this example, we use the `$GEMSTONE` installation tree to avoid having to duplicate files that can be shared. To reduce I/O contention, we create a separate data directory on another disk.

Step 1. Copy a fresh repository extent and configuration file to the new data directory. Make the files writable by the development group.

```
% mkdir /user2/devdata
% cd /user2/devdata
% cp $GEMSTONE/bin/extent0.dbf .
% cp $GEMSTONE/bin/initial.config system.conf
% chmod ug+w extent0.dbf system.conf
```

Step 2. Edit the new configuration file so that it specifies the proper extent file. Change the transaction log directories to the new data directory.

```
DBF_EXTENT_NAMES = /user2/devdata/extent0.dbf;
STN_TRAN_LOG_DIRECTORIES = /user2/devdata,
/user2/devdata;
```

Step 3. Set the environment variable `GEMSTONE_SYS_CONF` so it points to the new configuration file. GemStone will use the new configuration file in place of the default, which is `$GEMSTONE/data/system.conf`. For example:

```
$ GEMSTONE_SYS_CONF=/user2/devdata/system.conf
$ export GEMSTONE_SYS_CONF
```

Step 4. Start the Stone for the development repository, giving it the name `devserver22`. GemStone will create log files with that server name as the prefix.

```
$ startstone devserver22
```

Step 5. Start linked Topaz, then set the GemStone name to `devserver22` and log in as DataCurator:

```
% topaz -l
topaz> set gemstone devserver22
topaz> set username DataCurator
topaz> login
GemStone Password?
successful login
topaz 1>
```

At this point, you are logged in much as during the initial installation and you can begin installing user accounts for developers. However, the repository is the one in `devdata`. Any changes that you commit to this repository will not affect

`$(GEMSTONE)/data/extent0.dbf`, and existing applications can use the latter repository independently.

1.7 How to Operate a Duplicate Server

Some customers may want to keep a duplicate of a production GemStone server running almost in parallel as a “warm” backup. The duplicate continually runs in restore mode, restoring each transaction log from the production server after the log is closed. This section tells how to set up the duplicate server and restore the logs.

For general information about restoring backups and transaction logs, see “How to Restore from a Smalltalk Full Backup” on page 9-14. This discussion assumes you are familiar with that procedure.

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 the transaction logs created by the duplicate server. You can do that by using different log directories or different file name prefixes.

Step 1. Install the duplicate server. For fault tolerance, it’s best to do a complete GemStone installation on a second node.

Step 2. While the production Stone is shut down, make an operating system copy of the production extents to the appropriate locations on the duplicate server.

Step 3. Start the duplicate server using the **-R** option, which causes the Stone to enter restore mode. For instance, **startstone -R**.

Step 4. 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 5. On the duplicate server, tell the Stone where to find the archive logs by sending the following message:

```
Repository>>setArchiveLogDirectories: arrayOfDirectorySpecs
      tranlogPrefix: tranlogPrefixString
```

The arguments specify the directories (or raw partitions) to which the production logs will be copied and the log prefix. Unless they will be changed during the copy operation, these parameters typically correspond to the

following configuration parameters for the *production* Stone: STN_TRAN_LOG_DIRECTORIES and STN_TRAN_LOG_PREFIX. For details, see the method comments in the image.

The settings continue in effect until the Stone is shut down.

The following example uses the names from Step 4:

```
topaz 1> printit
SystemRepository setArchiveLogDirectories:
    #( '$GEMSTONE/data' )
tranlogPrefix: 'prodtranlog'
%
```

Step 6. On the production server, periodically send `Repository>>startNewLog` to force a new transaction log. Copy the previous log to the duplicate server, being careful to use the destination directory or file name prefix selected in Step 4.

Alternatively, you can use `Repository>>currentLogFile` to monitor the current transaction log in use on the production server. When the log file changes, copy the recently completed transaction log to the duplicate server.

Step 7. As each log copy arrives on the node where the duplicate runs, restore it using `Repository>>restoreFromArchiveLogs`. Each time this method is invoked, it restores any new logs that it finds using the information provided by `Repository>>setArchiveLogDirectories:tranlogPrefix:` (Step 5).

Step 8. Repeat Steps 6 and 7 as necessary.

Step 9. If it becomes necessary to activate the duplicate in place of the production server, first restore any remaining transaction logs from the production server. Then, on the duplicate server, send the message `Repository>>commitRestore` to terminate the restore process and enable logins.

—
|

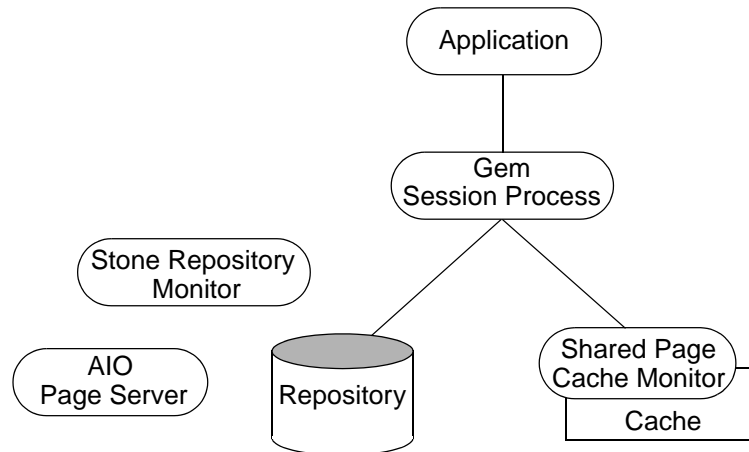
Configuring Gem Session Processes

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 3.

2.1 Overview

As shown in Figure 2.1, a GemStone session involves the following components in a client-server relationship:

- The user application
- A session manager process (Gem), which acts as a server for a particular application
- The Stone repository monitor
- The shared page cache monitor and cache
- The Stone's AIO page server
- The repository itself

Figure 2.1 GemStone Session Elements

The Gem session process provides the bulk of the repository capabilities as seen by the application. From the viewpoint of the application, the Gem *is* the object server:

- It logs in to the repository through the Stone repository monitor, and it obtains object locks, free object identifiers, and free pages from the repository monitor.
- 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 (as in Figure 2.1) or integrated with the application into a single process, in which case the application is called a *linked* application.

When the Gem runs as a separate process, it responds to Remote Procedure Calls (RPCs) from the application, in which case the application 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.

GemStone provides both linked and RPC tools for repository administration. GemStone also provides both types of libraries for application developers. RPC applications start the Gem session process as part of connecting a user to the repository.

NOTE

Whether an application is linked or RPC depends on which GemStone library was loaded at run time. Either type of application can be used on a single node or across a network. Only one session can be linked, but the application can have multiple RPC sessions. C programmers should use an RPC version during development and debugging to protect Gem data structures from possible corruption.

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 sought by the repository monitor when it starts. However, there are three important differences:

- The session configuration file is optional. If one 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 repository monitor by network protocol and are the same for all sessions logged in to that Stone.
- The first session process on a node remote from the Stone and extents uses the shared page cache configuration options (SHR_), which determine the configuration of the cache on that node.

Sometimes it's useful for certain sessions to use a variant configuration. Appendix A, “GemStone Configuration Options,” tells how to specify an alternate configuration file and how to use supplementary files to adjust the system-wide configuration for a specific session process. Appendix A describes each of the configuration options.

2.2 How to Configure Gem Session Processes

To configuring a Gem session process, you perform the following steps:

1. Gather application specifics about the number of sessions that will be logged in to the repository simultaneously from this node.
2. Plan the operating system resources that will be needed: memory and swap (paging) space.
3. Set the Gem configuration options.

If this node is remote from the repository monitor, enable a local GemStone shared page cache. Gem session processes running on a server node always use the Stone's shared page cache.

4. Set GemStone file permissions to allow session processes access while providing adequate security.

Gathering Application Information

System resources needed for session processes primarily depend on the number of sessions that will be logged in to a particular repository from this node. Remember that in some applications each user can have more than one session logged in.

Planning Operating System Resources

GemStone session processes need adequate memory and swap space to run efficiently. In addition, kernel parameters can limit the number of sessions that can connect to the shared page cache.

Estimating Memory Needs

Two factors determine the memory needs for session processes:

- The size of the shared page cache on a node remote from the Stone and extents will depend on the configuration of the Gem that starts the cache. (There is only one cache on each node for a particular repository; session processes running on the server node attach to the Stone repository monitor's cache.)
- 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.

Assuming the default of 10 MB for the Gem's temporary object cache, the first Gem session process on a node ordinarily requires about 30 MB of memory, of which 5 MB is for code that can be shared by other session processes. Each additional session process requires about 25 MB. The requirement is the same for Gems linked with an application. If you tune the cache size for Gems (page 2-11), add any increase to the amount given here.

In addition to the memory needs for session processes, you must also allocate memory for object server processes. For details, see "Planning Operating System Resources" on page 1-12.

For Gem session processes running on machines that are remote from the object server, there are additional memory needs on the server. For information about this, see "Estimating Memory Needs" on page 1-12.

Estimating Swap Space Needs

Swap (paging) space on machines remote from the Stones should follow the same general guidelines given for servers on page 1-13. In determining the additional swap space needed for GemStone session processes, use the memory requirements derived in the preceding section ("Estimating Memory Needs"), including space for the number of sessions you expect. The resulting figures will approximate the client's needs, and are in addition to the swap requirement for the object server and non-GemStone processes.

Estimating File Descriptor Needs

When a Gem session process starts, it attempts to raise the file descriptor limit from the default (soft) limit to the hard limit set by the operating system. GemBuilder applications (both linked and RPC) and page servers do the same. Gem session processes use file descriptors this way:

- 7 for stdin, stdout, stderr, and internal communication
- 2 for a connection between the Gem and an RPC application
- 1 for each local extent within a file system
- 2 for each local extent that is a raw partition
- 1 for each extent on a remote node

GemBuilder applications that start a large number of RPC Gems need a correspondingly large number of file descriptors.

You can override the default behavior of raising the file descriptor limit to the hard limit by setting the GEMSTONE_MAX_FD environment variable to a positive integer. A lower limit may be desirable in some cases to reduce the amount of

virtual memory used by the process. A value of 0 disables attempts to raise the default limit.

The value of `GEMSTONE_MAX_FD` in the environment of a NetLDI (Network Long Distance Information) server is passed to its child processes.

Reviewing Kernel Tunable Parameters

The kernel parameter of primary relevance to GemStone session processes is the maximum number of semaphores per semaphore id (typically `semmsl` or a similar name, although it is not tunable under all operating systems). This parameter limits the number of sessions than can connect to the shared page cache, because each session uses one semaphore.

How you determine the existing limits depends on your operating system. If the information is not readily available, proceed anyway. A later step shows how to verify that the limits are adequate for the GemStone configuration you set up.

Set the Gem Configuration Options

Initially, you should focus on configuration options for the shared page cache. Changes to other Gem configuration options are discussed later in this chapter, beginning on page 2-11.

The Stone repository monitor always creates a shared page cache monitor and cache on its node and on any other node that contains an extent. These caches are based on parameters in the Stone's configuration file.

On other nodes, when the first Gem session process logs in, the Stone starts a shared page cache monitor and cache on that node. Parameters in the Gem's configuration file 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). All subsequent sessions that log in from that remote node will be directed to use the same cache.

You can use the GemStone `shmем` utility (described on page 1-18) to determine whether the kernel configuration on a node remote from the Stone is adequate to support the cache. Use arguments from the configuration file that will be read by Gems running on that node.

To Set Ownership and Permissions for Session Processes

The primary consideration in setting file ownership permissions for client access is to make sure the Gem session process can read and write both the extents and the shared page cache. You must also take into account the following factors:

- Is the Gem session process linked with the application or RPC?
- Does the Gem session process runs on the server or on a node remote from the Stone?
- Does the server uses setuid bit and protection mode 600 for the extents (as recommended on page 1-31), or does it use the alternative of group write permission?

Extents

The extents may be protected as read-write only by their owner (protection 600) if you use the setuid (S) bit for repository executables as recommended on page 1-31. Otherwise, the extents must be writable by a group to which the GemStone users belong (protection 660).

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 is read-write for the owner and group (the equivalent of file protection 660). You can inspect the cache ownership and permissions by using the `ipcs` command. (These permissions are not configurable by users.)

For a session to log in using a shared page cache, the UNIX 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 3, “Connecting Distributed Systems.”

If the setuid bit is set on repository executables as recommended in Table 1.2 on page 1-31, the Stone process and shared page cache will belong to the owner you specify for those files (such as `gsadmin`).

To Set Access for Linked Applications

For linked applications *on the server*, we recommend you try using the setuid bit on the application’s executable file. Have the file owned by `gsadmin` as it is defined on page 1-31. This works well for `topaz -l`. The `installgs` script offers to set the file ownership and permissions for you. To do it manually, do this while logged in as root:

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

You may prefer not to use the `setuid` bit with linked applications that do not distinguish between real and effective user IDs. 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). Linked applications that do not make this distinction, such as a third-party Smalltalk used with GemBuilder, are likely to perform *all* I/O as the effective user, or *gsadmin*. If this result is unsatisfactory, remove the `S` bit on that executable and add group write permission to the extents.

To Set Access for All Other Applications

All applications except linked applications on the server always use a GemStone NetLDI service to start a separate Gem session process or, in some cases, a page server. For these sessions, we recommend that the Gem session process and page server always be owned by (run as) the *gsadmin* account. That arrangement ensures that the Gem will be able to read and write both the extents and the shared page cache. The ownership and protection of the application executables themselves is not a factor.

To Set Access to Other Files

GemStone creates log files and other special files for session processes in several locations, which are described below. 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.

`$HOME` GemStone ordinarily creates log files for spawned processes (such as RPC Gem session processes and page servers) in the home directory of the user or the NetLDI captive account. In situations where the home directory cannot be writable, the environment variable `GEMSTONE_NRS_ALL` can be used to specify an alternative location; see "To Set a Default NRS" on page 3-15.

`/opt/gemstone` All users should have read/write/execute access to the directories `/opt/gemstone/log` and `/opt/gemstone/locks` on each host. (On some systems, these directories may be located in `/usr/gemstone`.)

By default, NetLDIs (Network Long Distance Information processes) create log files in the `log` directory.

GemStone processes that have a name for each instance (currently the Stone repository monitor, shared page cache monitor, and NetLDI) create lock files in the `locks` directory.

2.3 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

Class methods in category `Configuration File Access` 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` in category `Runtime Configuration Access` lets you change the value of the internal run-time parameters in Table 2.1 if you have the appropriate privileges. The parameters that can be changed are those for which

`configurationParameterDict: aName` returns a negative `SmallInteger`. All changeable parameters require that *aValue* be a `SmallInteger`.

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 2.1 Session Configuration Parameters Changeable at Run Time

Configuration File Option	Internal Parameter
GEM_FREE_FRAME_LIMIT	#GemFreeFrameLimit
GEM_IO_LIMIT	#GemIOLimit
GEM_PGSRV_UPDATE_CACHE_ON_READ	#GemPgsvrUpdateCacheOnRead

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.”

2.4 How to Tune Session Performance

There are a number of configuration options by which you can tune your Gem session processes. These options can help make better use of the Gem's internal caches, reduce swapping, and control disk activity limiting the I/O rate for certain sessions.

To Tune the Temporary Object Space

You should increase `GEM_TEMPOBJ_CACHE_SIZE` (page A-19) for applications that create a large number of temporary objects — for example, applications that make heavy use of the reduced conflict classes or sessions performing a bulk load.

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. This is particularly likely to be a problem if there are long transactions that modify a large number of objects.

You will probably need to experiment somewhat before you determine the optimum size of the temporary object space for the application. The default of 10000 (10 MB) should be adequate for normal user sessions. For sessions that place a high demand on the temporary object cache, such as upgrade, you may wish to use 100000 (i.e., 100 MB).

The following Class methods in class System are available to help you keep track of the load on temporary object memory:

`_tempObjSpaceUsed`

Returns the approximate number of bytes of temporary object memory being used to store objects.

`_tempObjSpaceMax`

Returns the approximate maximum number of bytes of temporary object memory which is usable for storing objects.

`_tempObjSpacePercentUsed`

Returns the approximate percentage of temporary object memory that is being used to store temporary objects. This is equivalent to the expression:

```
(System _tempObjSpaceUsed * 100) // System
_tempObjSpaceMax.
```

It is possible for the result to be slightly greater than 100%; this indicates that temporary memory is almost completely full.

Any increase in `GEM_TEMPOBJ_CACHE_SIZE` translates directly into increased memory usage per user.

To Tune the Private Page Cache

The configuration option `GEM_PRIVATE_PAGE_CACHE_KB` sets 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.

You can use VSD to monitor the value of the statistic `LocalCacheOverflowCount` (page 8-36). If that value is non-zero, you can increase `GEM_PRIVATE_PAGE_CACHE_KB` as needed. For details, see page A-17.

To Reduce Excessive Swapping of Sleeping Sessions

Excessive swapping can be caused by the need to awaken (and swap in) sleeping sessions that are outside of a transaction. When the Stone runs out of space in the shared page cache, it takes this action (by sending a `SignaledAbort` message) to store the old commit records on which the sleeping sessions are based. Each such session must awaken long enough to update its view of the repository.

It may be possible to reduce this type of swapping by changing the server configuration. See the discussion and procedure on page 1-41.

2.5 How to Install a Custom Gem

The *GemBuilder for C* manual explains how to create a custom Gem session executable containing your own C functions to be called from Smalltalk. One way to make this custom Gem available to all users is to perform the following steps as system administrator:

Step 1. Copy the shell script `gemnetobject` from `$GEMSTONE/sys` to your working directory. This shell script is used to start Gem session processes under the UNIX shell. You will modify this script to start your custom Gem executable instead of the standard one.

Step 2. In your copy of `gemnetobject`, find the section labeled `User-definable symbols`. In that section, replace `gem` in the line

```
gemname="gem"
```

with the name of the new Gem executable. For example:

```
gemname="MyGem"
```

Step 3. Rename your modified copy of the shell script `gemnetobject` so that it has a distinct filename. For example:

```
% mv gemnetobject MyGemnetObject
```

Step 4. Copy the new shell script to `$GEMSTONE/sys`. Make sure that all GemStone users have read and execute (**r-x**) permission for the script. For example:

```
-r-xr-xr-x 1 root 912 Jul 24 20:22 MyGemnetObject
```

If necessary, change the permissions:

```
% chmod 555 MyGemnetObject
```

Step 5. Add an entry for the new shell script to the services database, `$GEMSTONE/bin/services.dat`. A NetLDI checks that file to translate the name of a service to a command it can execute. For example:

```
MyGemnetObject $GEMSTONE/sys/MyGemnetObject
```

Step 6. Copy the new Gem executable to the GemStone system directory. For example:

```
% cp MyGem $GEMSTONE/sys
```

Step 7. Make sure that all GemStone users have read and execute (**r-x**) permission for the new Gem executable.

The custom Gem executable is now available for shared use.

—
|

Connecting Distributed Systems

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

- *Overview* (page 3-2) — An introduction to the GemStone processes and network objects that facilitate distributed GemStone systems.
- *How to Arrange Network Security* (page 3-9)— Three ways to provide access to GemStone processes on other nodes.
- *How to Use Network Resource Strings* (page 3-15) — How to specify where distributed GemStone resources are located.
- *How to Set Up a Remote Session* (page 3-18) — Step-by-step examples for setting up typical distributed client-server configurations. It also contains troubleshooting tips.

3.1 Overview

A properly configured network system is nearly transparent to GemStone users, but it requires additional steps by the system administrator. Users must be given access to all the workstations that will run their GemStone processes. Pointers to network services must be set up, and file and process specifications must include the node name in addition to the file name and path. Because processes are running on different nodes, the log files are spread throughout the network, and troubleshooting may become more complicated.

The nodes in your system can be any combination of GemStone-supported platforms, as long as they are connected by means of TCP/IP. Each remote GemStone connection consists of two TCP/IP connections to compensate for out-of-band problems in TCP/IP. (For remote Gems that are configured with `GEM_PGSRV_UPDATE_CACHE_ON_READ` set to true, only one TCP/IP connection is open.)

Although the Sun Network File System (NFS) can be used to share executables, libraries, and configuration files, they are not required and are never used to share repository files. Instead, GemStone extends the capabilities of TCP/IP by adding special network servers and page servers, which are described later in this section.

Figure 3.1 and Figure 3.2 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.

In Figure 3.1, 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 can access the shared page cache that was started by the Stone repository monitor. For instructions on setting up this configuration, see “To Run the Gem Session Process on the Stone’s Node” on page 3-23.

In Figure 3.2, the application and the Gem are linked in a single process that runs on the remote node. This configuration avoids the overhead of RPC calls, but in some applications it may increase network traffic substantially if large objects must be brought across the network. The Stone repository monitor starts a shared page cache on the remote node when the first user from that node logs in to the repository. The Stone and the Gem session process each use a GemStone page server to access data pages residing on the other node. For instructions on setting up this configuration, see “To Run a Linked Application on a Remote Node” on page 3-20.

Figure 3.1 Gem Session Process on Server Node

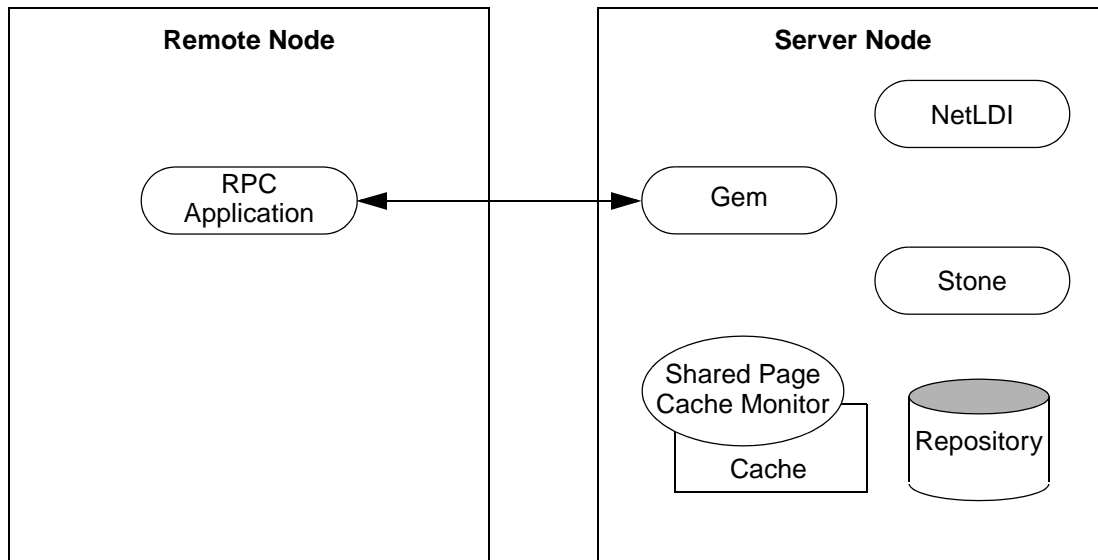
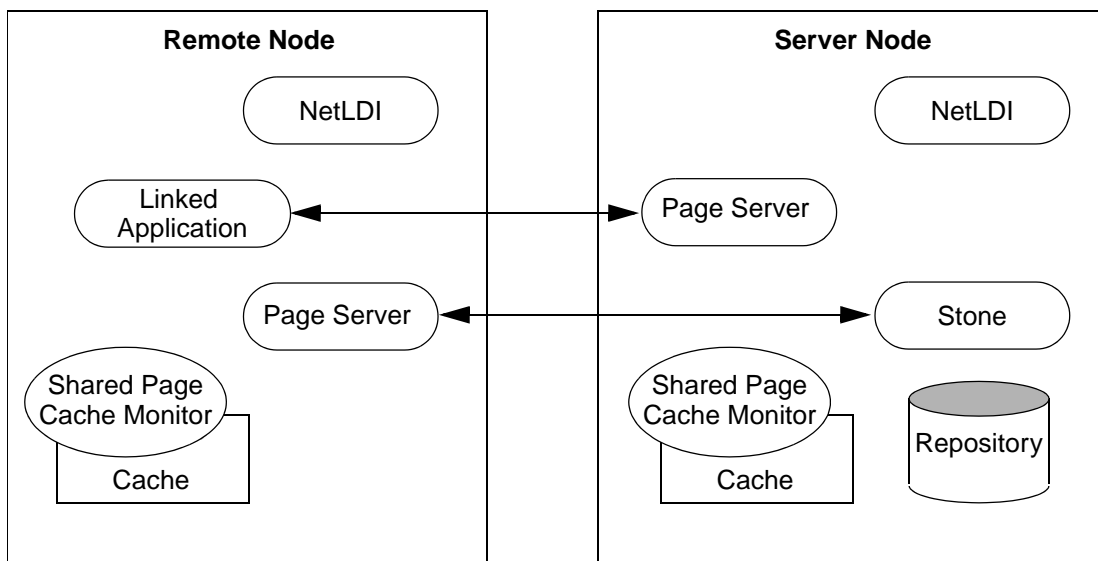


Figure 3.2 Gem Session Process on Remote Node



GemStone NetLDIs

The GemStone network server process is called NetLDI (Network Long Distance Information). The NetLDIs are the glue holding a distributed GemStone system together. Each NetLDI reports the location of GemStone services on its node to remote processes that must connect to those services. It also spawns other GemStone processes on request.

In a distributed system, each node where a Stone repository monitor, Gem session process, or linked application runs must have its own NetLDI. (That is, you do not need a NetLDI on nodes where only RPC applications are running.)

You start a NetLDI directly by invoking the **startnetldi** command (page B-13). The NetLDI, in turn, starts Gem session processes and page servers on demand. (See the following section for more about page servers.) These child processes belong by default to the user account of the process requesting the service—sometimes that account is a user logging in to GemStone, other times it is the account that started the repository monitor.

Because most operating systems only let the root account start processes that will be owned by other accounts, a NetLDI ordinarily must run as root if it is to serve more than one user. This ownership can be accomplished by setting the owner and S bit for `$/GEMSTONE/sys/netldid` or by starting the NetLDI while logged in as root. For information about the S bit, see “To Set File Permissions for the Server” on page 1-30. For the default installation, the file permissions and ownership for the NetLDI executable should look like this:

```
-r-sr-xr-x 1 root gsadmin 516096 Jul 29 22:01 netldid
```

To map a GemStone service name (such as a Stone name) to a network port number, GemStone checks for a lock file named *serviceName.LCK*, in the directory `/opt/gemstone/locks` or `/usr/gemstone/locks`. Every Stone, NetLDI, and shared page cache monitor creates one of these files when it starts. If there is no lock file and the service is a NetLDI, GemStone then checks for an entry in `/etc/services`, the TCP/IP network database. That file must contain an entry giving the port number for the NetLDI.

NetLDI Names

The default name of the NetLDI process is `gs64ldi`. During installation, this name is added to the `/etc/services` file and assigned a port number. You can change the name by using **startnetldi** *netLdiName*. The name may contain digits, but it must not be entirely numeric. If you use a different name, also do the following:

- Add the new name and a port number to `/etc/services`. If you have a distributed GemStone system, make the same entry on each node.
- Set `#netldi` to `netLdiName` in the `GEMSTONE_NRS_ALL` environment variable for each user. For example,

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

For more information about `GEMSTONE_NRS_ALL`, see “To Set a Default NRS” on page 3-15.

GemStone Page Servers

Remote GemStone repository I/O is carried out by page server processes. The name of the executable file is `pgsvrmain`. For each process that connects to a repository extent across the network (that is, for the repository monitor and each session process), the NetLDI service spawns a `pgsvrmain` on the node where the extent resides. GemStone never uses NFS (network file system) for repository access.

The Stone repository monitor uses a page server to perform asynchronous I/O to the repository. This page server is created at start up and is present even if all GemStone sessions are local.

You can also start additional AIO page servers as well as page servers dedicated to a single task—adding free frames to the free frame list. For details and instructions, see “Adding Page Servers” on page 1-43.

If you have many different extents on different spindles, starting more page servers can improve performance.

GemStone Network Objects

GemStone uses the concept of *network objects* to encompass the services that a NetLDI can provide to a client. In addition to the page server, other network objects include the following services requested by the Stone at startup: the shared page cache monitor, SymbolGem, page manager, and one or more garbage collection (GcGem) sessions.

The network object most visible to users is the Gem session process requested by an RPC application. This object can be `gemnetobject` or the name of a custom Gem. The request can be sent to the NetLDI on the same node to start a local session process, or (by using a network resource string) the request can be sent to a NetLDI on another node to start a process there.

The NetLDI first tries to map the requested object to the path of an executable by looking for an entry in `$GEMSTONE/bin/services.dat`. There is 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 maps the request to the script `$GEMSTONE/bin/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()`.

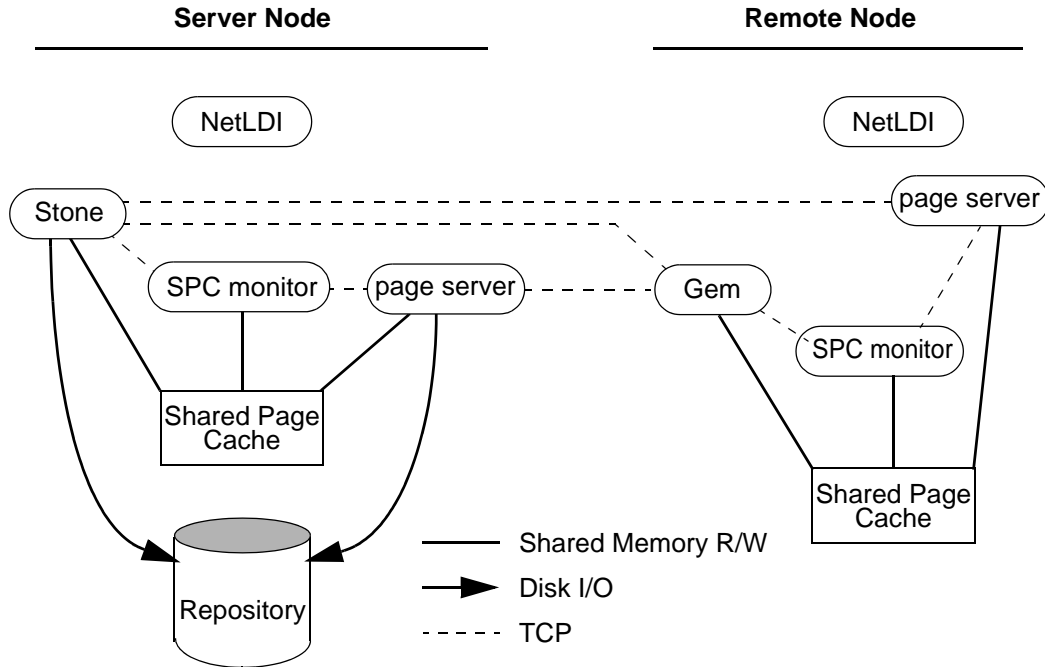
If your application uses a custom Gem executable, you can edit `services.dat` to include the appropriate mapping. For the procedure, see “How to Install a Custom Gem” on page 2-12.

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.

Shared Page Cache in Distributed Systems

When the remote session logs in to the repository, the Stone repository monitor uses a NetLDI and page server (`pgsvrmain`) on the remote node to start a monitor process, and that monitor uses the NetLDI to create a local shared page cache. When the remote Gem wants to access a page in the repository, it first checks the shared page cache on the remote node. If the page is not found, the Gem uses a `pgsvrmain` on the server node, checking in the shared cache on that node and then, if necessary, reading the page from the disk. See Figure 3.3.

Figure 3.3 Shared Page Cache with Remote Gem



Disrupted Communications

Several incidents can disrupt communications between the GemStone server and remote nodes in a distributed configuration. Examples include node crashes and loss of the communications 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.

Your operating system documentation contains information about the TCP keepalive option.

NOTE

Changes to this option on a given node affect all network communications on that node.

Solaris (Sun):

```
/usr/sbin/ndd -set /dev/tcp tcp_keepalive_interval value
```

HP-UX:

```
/usr/contrib/bin/nettune -s tcp_keepstart value
```

AIX:

```
/etc/no -o tcp_keepidle=value
```

Linux:

```
/sbin/sysctl -w net.ipv4.tcp_keepalive_time=value
```

3.2 How to Arrange Network Security

This section describes the levels to which the system administrator can set the GemStone authentication requirement.

Default

In the default NetLDI mode, authentication is required each time a NetLDI attempts to start certain processes for a client, even if that process is to run on the node where the user is logged in. These situations always require authentication:

- Starting an RPC Gem session process, even on the same node.
- Creating or restoring a GemStone backup using a device on a node remote from the Gem performing the operation.
- Using **copydbf** between nodes.

Once a Stone or Gem is running, the NetLDI treats it as a trusted client and starts the page servers needed by a remote login without authentication. Simple network information requests, such as a request to look up a port number, also do not require authentication.

Secure Mode

startnetldi -s starts the NetLDI in *secure mode*. All accesses are authenticated, including simple requests to look up a server name. This mode affects the **waitstone** command and such user actions as connecting a session process to a remote Stone (a NetLDI is asked to look up the Stone's address).

NOTE

Secure mode requires authentication before a Gem or Stone can start a page server to access an extent or shared page cache on another node. Under this mode, the account that starts the Stone process may need an entry in the account's .netrc file for each node in the GemStone system, and GemStone user accounts may need a .netrc entry for each node on which the extents are located.

Captive Account Mode

startnetldi -aname starts the NetLDI in *captive account mode*, which provides additional security. All child processes created by the NetLDI will belong to the single, designated account *name*. This mode requires that the NetLDI run either as root or as *name*. The effect is much like setting the S bits on executables, but it only affects ownership of processes started by the NetLDI, not linked applications

invoked directly by the user. Because this mode by itself does not change the authentication requirement, on most systems the NetLDI must either run as root so that it can authenticate other users or run in guest mode, which suspends authentication. For more information, see “Alternative: Guest Mode With a Captive Account” on page 3-14.

The captive account can be an ordinary user account or one created for that purpose, such as a GemStone administrative account. Log files by default will be in the captive account’s home directory. Although captive accounts provide access to the repository, they do not affect network access—if authentication is required, it is based on the identity of the real user who requests the service.

Guest Mode

startnetldi -g starts the NetLDI in *guest mode*. No accesses are authenticated. When it is used by itself, guest mode lets the user who started the NetLDI also start other GemStone processes without typing passwords or creating `.netrc` files. Because guest mode is not permitted if the NetLDI will run as root, guest mode usually is combined with captive account mode. Table 3.1 shows how guest mode and captive account mode affect NetLDI operation.

Table 3.1 NetLDI Guest Mode and Captive Account Mode

NetLDI Options	Passwords Required	Owner of Spawned Processes	Owner of NetLDI Process	Which Accounts Can Start Processes
(none)	Yes, for RPC Gem or copydbf between nodes	Client's account	Ordinary user	Owner of NetLDI
			Root	Any user
-aname	Yes, for RPC Gem or copydbf between nodes	Account <i>name</i> (must start the NetLDI)	Ordinary user (<i>name</i>)	Owner of NetLDI
			Root	Any user
-g	No	Client's account	Ordinary user	Owner of NetLDI
			Root—not allowed	
-aname -g	No	Account <i>name</i> (must start the NetLDI)	Ordinary user (<i>name</i>)	Any user
			Root—not allowed	

For a complete list of the **startnetldi** command-line options, see page B-13.

The following topics describe ways of setting up authentication to serve multiple users:

- Password authentication (the default) with the NetLDI running as root (page 3-12)
- Guest mode combined with a captive account (page 3-14)

Examples later in this chapter include procedures for specific configurations (see “Configuration Examples” on page 3-19).

Default: Password Authentication

The GemStone default is to use a system login name and password to authenticate network access. There are several ways for the user to provide this information:

- Create a `.netrc` file containing the name of the other node, the login name, and the password. (See “Using a `.netrc` File” on page 3-12.)
- Enter the login name and password through the application’s user interface, such as the `HostUserName` and `HostPassword` parameters in `Topaz`. (See “Using the Application Interface” on page 3-13.)
- Use the NRS authorization modifier `#auth:loginName@password` as part of a process name or file name. (See “Using an NRS `#auth` modifier” on page 3-13.)

If the user does not provide the login name and password explicitly, the application or GemStone executable tries to read them from a `.netrc` file in the user’s home directory.

NOTE

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

The NetLDI providing the service verifies the password against the entry in the password file (or Network Information Service). Under operating systems that support shadow password files, the NetLDI first checks the shadow file; if it finds an entry, it uses that entry in preference to the entry in `/etc/passwd`.

For the default installation, the file permissions and ownership for the NetLDI executable should look like this:

```
-r-sr-xr-x 1 root gsadmin 516096 Jul 29 22:01 netldi
```

Using a `.netrc` File

Create a `.netrc` file in the home directory of each user who will be doing any of the following:

- Running an RPC Gem session process.
- Creating or restoring a GemStone backup using a device on a node remote from the Gem performing the operation.
- Running `copydbf` between nodes.

If the user has a home directory on more than one node, the easiest way is to make a file containing an entry for each node and install a copy in all of the home

directories. The file must contain the login information for each node where that user will need an RPC Gem or a page server.

GemStone supports the basic `.netrc` options of `node`, `login`, and `password` (which must appear in that order). For each node, the `.netrc` file should contain one line like the following:

```
machine nodeName login systemLogin password userPassword
```

NOTE

The node name in the .netrc file must exactly match the name as it is listed in DBF_EXTENT_NAMES or as provided to an application as a login parameter. In particular, any domain qualification must be the same.

Because the `.netrc` contains hard-coded passwords, it should be protected in such a way as to be readable only by its owner.

Using the Application Interface

Your application's login interface may let you specify a node 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 as variables:

```
topaz> set hostusername yourLogin
topaz> set hostpassword yourPassword
```

GemBuilder for Smalltalk provides similar fields in its login dialog: **Host username** and **Host password**.

Using an NRS #auth modifier

A third way to specify the login name and password is to provide that information in NRS syntax as part of the name of a process that NetLDI is to start. Ordinarily, an application program provides the name and password using information obtained from the user. For example, if you set the Topaz login parameters `HostUserName` and `HostPassword`, the application puts them in an NRS like the following:

```
'!tcp@Server#auth:HostUserName@HostPassword!gemnetobject'
```

The GemStone C interface provides similar capability to application programmers. For further information, refer to calls described in the *GemBuilder for C* manual.

Although it is less convenient for ordinary use, administrators and programmers may find it helpful in testing to enter the authorization modifier directly using the Topaz *GemNetId* parameter. For example:

```
topaz> set gemnetid !@Server#auth:name@password!gemnetobject
```

Alternative: Guest Mode With a Captive Account

The NetLDI guest mode can best be combined with captive account mode (page 3-9) in which a single, designated account owns all processes spawned by the NetLDI. The result serves multiple users with the convenience of guest mode and with improved security because the child processes no longer belong to accounts of individual users who request services.

The principal advantage of this combination is that the NetLDI can spawn processes on behalf of multiple users without being run as root. To make this capability possible, the captive account must own the `netldi` process. Change the file permissions and ownership for the NetLDI executable to remove the S bit:

```
-r-xr-xr-x 1 gsadmin gsadmin 516096 Jul 29 22:01 netldi
```

A disadvantage of the captive account for some applications is that the Gem session process will perform *all* I/O as that account, not as the account running the application — all file-ins, file-outs, and `System class>>performOnServer:`.

The captive account mode differs from the `setuid` method (page 1-31) in that 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/bin/services.dat`.) If you prefer, you can prohibit such ad hoc services by specifying the `-n` option when starting the NetLDI.

If the combination of guest and captive account modes fits your needs, follow this configuration procedure:

- Step 1.** Create a UNIX account to own the GemStone distribution tree and serve as the captive account. We will refer to this account as *gsadmin*.
- Step 2.** Make *gsadmin* the owner of the tree, and set the `setuid` bit for any linked GemStone executables that run on the server node. Make the repository extents accessible only by *gsadmin* (mode 600). For instructions, see “To Set File Permissions for the Server” on page 1-30.

Step 3. Make sure *gsadmin* has execute permission for `$/GEMSTONE/sys/netltdid`. The `setgid` bit should NOT be set on the `netltdid` executable. For instance:

```
-r-xr-xr-x  1 gsadmin 516096 Jul 29 22:01 netltdid
```

Step 4. Log in as the captive account (such as *gsadmin*), then start the NetLTI in guest mode and captive account mode, and perhaps disallow ad hoc processes (the `-n` switch). For instance:

```
% startnetltdi -g -a gsadmin -n
```

For details about the `startnetltdi` command and its options, see page B-13.

3.3 How to Use Network Resource Strings

Once you have chosen the remote and server nodes, network resource strings (NRS) allow you to specify the location of each part of the GemStone system. Use an NRS on a network system where you would use a process or file name on a single-node system. For example, suppose you want to know whether a Stone is running. If the Stone is on the local node, use this command:

```
% waitstone gemStoneName -1
```

If the Stone is on a remote node, use a command like this instead:

```
$ waitstone !@oboe!gemStoneName -1
```

where *oboe* is the Stone's node. You can also use an Internet address in "dot" form, such as `120.0.0.4`, to identify the remote node. Note that each "!" must be preceded by a backslash (\) when your command will be processed by the C shell.

Appendix B, "GemStone Utility Commands," indicates which options of each UNIX-level GemStone command can be specified as an NRS. Besides location, an NRS can describe the network resource type so that GemStone can more accurately interpret the command line. Sometimes an NRS can also include your authorization to use that resource. For more information, see Appendix C, "Network Resource String Syntax."

To Set a Default NRS

You can set a default NRS header (the part between "!" ... "!") by setting the environment variable `GEMSTONE_NRS_ALL`. This variable determines which modifiers GemStone will use by default in each NRS it processes on your behalf.

For instance, you can cause all Gem session process logs to be created with a specific name in a specific directory.

- 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.
- 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 request.

Use the `#dir` modifier to set the current (working) directory for NetLDI child processes, such as `gemnetobject`. Without this setting, the default is the user's home directory. If the directory specified does not exist or is not writable at run time, an error is generated. For example:

```
$ GEMSTONE_NRS_ALL=#dir:/user2/apps/logs
$ export GEMSTONE_NRS_ALL
```

For further information about the modifiers and templates available, see Appendix C, "Network Resource String Syntax."

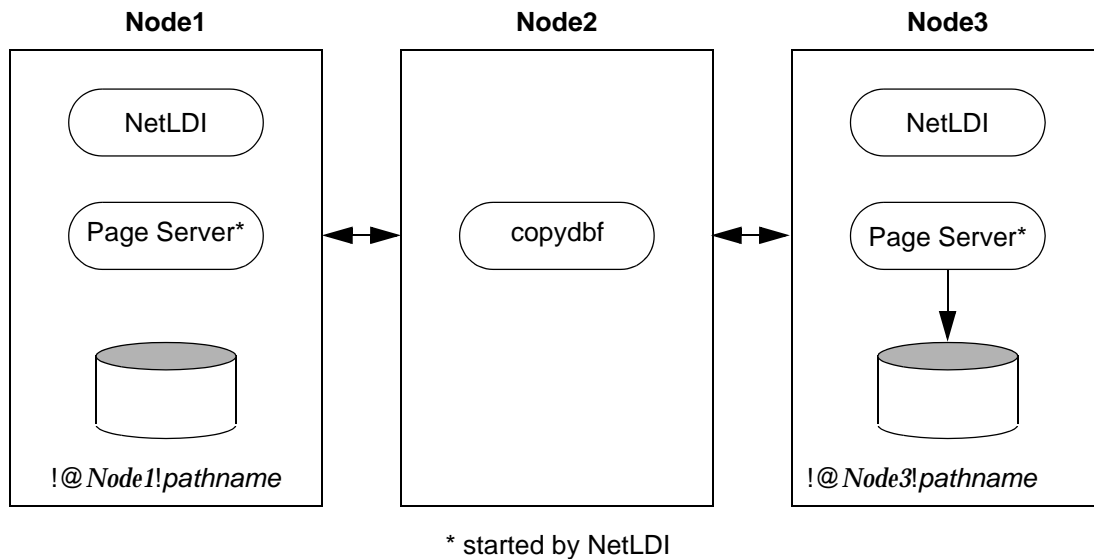
To Use `copydbf` Between Nodes

Figure 3.4 shows an application of `copydbf` in which the source and destination are on remote nodes (Node1 and Node3, respectively). NetLDIs and network access are required to spawn page servers on the two remote nodes.

NOTE

If you want to start a Gem on a remote machine, you need to have a NetLDI on both machines.

Figure 3.4 Connections for copydbf



Step 1. Unless the NetLDIs are running in guest mode, you will need to provide authorization for NetLDI services. Create a `.netrc` file in your home directory on Node2 containing a line like the following for each of the other nodes:

```
machine Node1 login userName password secret1
machine Node3 login userName password secret3
```

Step 2. If they are not already present, start NetLDIs on Node1 and Node3.

Step 3. When you issue the `copydbf` command, include the node names in NRS syntax and specify the full path. For example:

```
Node2% copydbf !@Node1!pathname !@Node3!pathname
```

3.4 How to Set Up a Remote Session

Configuring a Gem session process on a remote node is much the same as configuring a session process on the server, which is described in Chapter 2, “Configuring Gem Session Processes.” Keep the following points 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.
- If your site doesn’t run NIS, add each node in the GemStone network to `/etc/hosts`.
- If your site doesn’t run NIS, add the NetLDI entry to `/etc/services` on each node. Be sure to specify the same name and network port number each time.
- It’s best if each node has its own `/opt/gemstone/log` and `/opt/gemstone/locks` (or `/usr/gemstone/log` and `/usr/gemstone/locks`) directories. 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.
- Unless you run the NetLDIs in guest mode with a captive account, *all* users ordinarily must have an account on the primary server node and on any other node on which the repository extents reside. It’s easier if the account name is the same on each node.
- 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 *all* nodes where a Gem session process will run or where an extent will reside.

You can either repeat the installation from the GemStone distribution media, as described in the next topic, or mount the directory on the server node that contains `$GEMSTONE` (page 3-19). Although GemStone never uses NFS to access the repository files, it can use NFS to access other files.

To Duplicate the GemStone Installation

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 the two copies of the initial repository (`$GEMSTONE/data/extent0.dbf` and `$GEMSTONE/bin/extent0.dbf`) and the complete upgrade directory (`$GEMSTONE/upgrade`).

To Share a GemStone Directory

The following example prepares to run an application and Gem session process on a remote node using a shared software directory on the server. The `GEMSTONE` environment variable points to the shared installation directory, which is on the node *Server* and is NFS-mounted as `/Server/users/g64stone`.

Step 1. Set the `GEMSTONE` environment variable to point to the NFS-mounted installation directory, and then invoke `gemsetup`:

```
(Bourne or Korn shell)
$ GEMSTONE=/Server/users/g64stone
$ export GEMSTONE
$ . $GEMSTONE/bin/gemsetup.sh
```

```
(C shell)
Remote% setenv GEMSTONE /Server/users/g64stone
Remote% source $GEMSTONE/bin/gemsetup.csh
```

Step 2. If they do not already exist, create the GemStone `log` and `locks` directories on the local node. (NetLDIs use this log directory.) You may need to have a system administrator do this for you as root.

```
# cd /opt
# mkdir gemstone gemstone/log gemstone/locks
# chmod 777 gemstone gemstone/log gemstone/locks
```

Configuration Examples

GemStone supports several configurations in which the application communicates with the Gem session process by using remote procedure calls (RPCs). Although the calls to network routines inevitably are time-consuming, they are essential when the application runs on a different node from the Gem, and they are desirable during code development because they isolate the application and Gem address spaces.

Use of RPC configurations for production repositories should be based on careful analysis of system loads and network traffic to select the most efficient configuration for a particular application. The RPC configuration may be desirable

when the application accesses large or complex objects that would saturate the network if they were brought across it on a frequent basis.

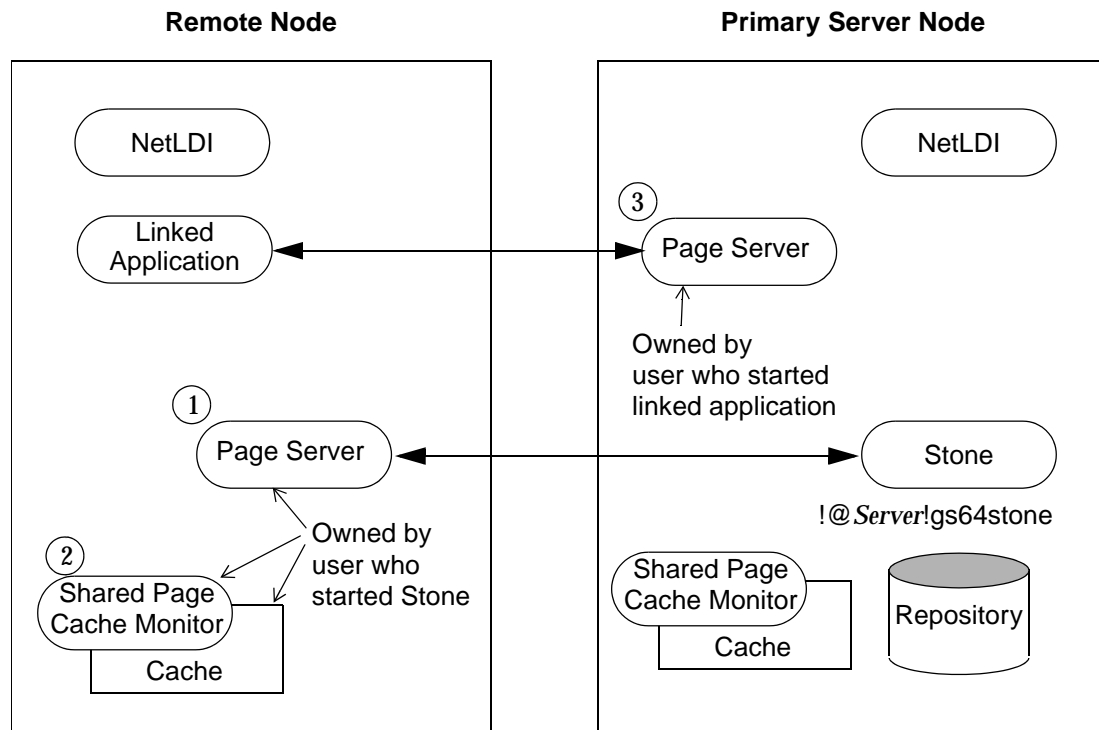
This section presents examples that illustrate the following distributed applications:

- A linked application connected to a Stone on another node (page 3-20).
- An RPC application with both the session process and the Stone on the server node (page 3-23).
- An RPC application with the session process on the application's node (page 3-25).
- An RPC application in which all three are on different nodes (page 3-28).

Two other examples show how to set up an extent on a node that is remote from the Stone, and how to use **copydbf** between nodes.

To Run a Linked Application on a Remote Node

Figure 3.5 shows how a linked application on a remote node communicates with a Stone and repository on the primary server node. This configuration typically is the best choice when you must offload some processes from a server node, especially when the application accesses relatively small objects or small groups of large objects.

Figure 3.5 Connecting a Linked Application to a Remote Server


Two NetLDIs and two page servers ordinarily are required. NetLDIs start the page servers on request of the Stone and the application. Numbers show the order in which these processes are started:

- One page server (1) lets the Stone start a shared page cache and monitor (2) on the remote node. The page server and monitor processes will be owned by the user who started the Stone (or by the captive account), so the owner must have an account on the remote node. The cache itself will have the same owner and group as the Stone. The linked application must have permission to access the cache, either through group membership or through an S bit on the application executable.
- The other page server (3) lets the Gem session process (the linked application) access the repository on the primary server. There will be one such page server

process on the primary server node for each session logged in from a remote node; its owner (which may be a captive account) must have an account on the primary server. The page server process must have read-write permission for the repository, either through group membership or through an S bit on the `pgsvrmain` executable.

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 “To Set Ownership and Permissions for Session Processes” on page 2-6.

The following steps set up a linked application on the remote node. They use software in an NFS-mounted installation on the primary server node; that directory is already mounted on the remote node as `/Server/g64stone`.

Step 1. Set the GEMSTONE environment variable to point to the installation directory, and then invoke `gemsetup`:

```
(Bourne or Korn shell)
$ GEMSTONE=/Server/g64stone
$ export GEMSTONE
$ . $GEMSTONE/bin/gemsetup.sh
```

```
(C shell)
Remote% setenv GEMSTONE /Server/g64stone
Remote% source $GEMSTONE/bin/gemsetup.csh
```

Step 2. Verify that a Stone and NetLDI are running on the primary server node. One way to do this verification is to use the `gslis`t utility. For example:

```
Remote% gslis -m serverName
```

(The `-m` option tells `gslis`t to list only processes that are running on the specified node. For more about `gslis`t, see page B-7.)

Step 3. Start a NetLDI on the remote node.

- To start the NetLDI for password authentication, make sure `$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):

```
Remote% startnetldi
```

- ❑ To start the NetLDI in guest mode (authentication is not required), make sure `$/GEMSTONE/sys/netldid` does NOT have the S bit set. Log in as the captive account *name*, then issue this command:

```
Remote% startnetldi -g -aname
```

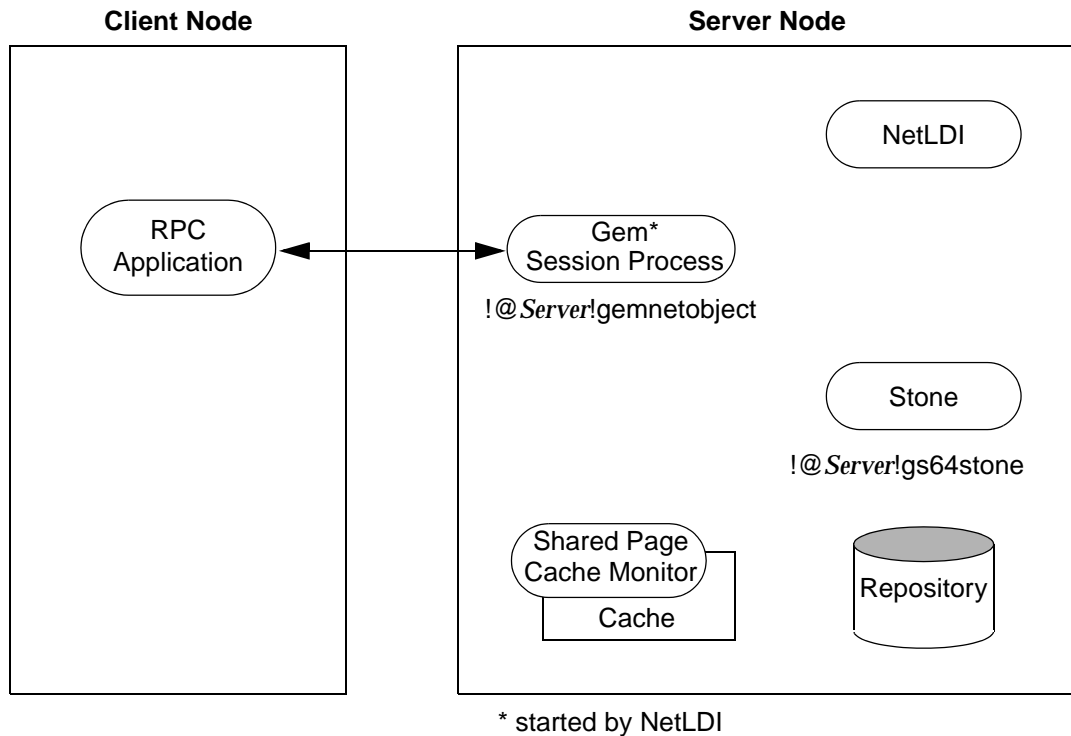
- Step 4.** Start the linked application (for instance, Topaz) on the remote node, then set the *GemStone* login parameter to include the name of the primary server node in network resource syntax. For instance, to log in to Topaz as DataCurator:

```
Remote% topaz -l
topaz> set gemstone !@Server!gs64stone
topaz> set username DataCurator
topaz> login
GemStone Password?
successful login
topaz 1>
```

To Run the Gem Session Process on the Stone's Node

If the Gem session process is going to run on the server node (as shown in Figure 3.6), 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. You should also specify the Gem network object (`gemnetobject`) that matches your UNIX shell on the server. For more information about network objects and how to invoke them, see “GemStone Network Objects” on page 3-5.

The following procedure assumes that you are already set up to run GemStone applications, as described in Chapter 2. In particular, you must have defined the `GEMSTONE` environment variable and invoked `$/GEMSTONE/bin/gemsetup` or its equivalent.

Figure 3.6 Starting a Session Process on the Server Node

Step 1. Make sure that the NetLDI and Stone are running on the server. One way to do this is to use the `gslist` command. For example:

```
Client% gslist -m serverName
```

Step 2. Unless the NetLDI is running in guest mode, decide how you will provide authentication.

- ❑ You can create a `.netrc` file in your home directory on the client node containing a line like the following, where the password is your password on the server:

```
machine Server login yourLogin password yourPassword
```

- Alternatively, you can set the application login parameters, such as `HostUserName` and `HostPassword`, after you start the application. For example:

```
topaz> set hostusername yourLogin
topaz> set hostpassword yourPassword
```

- Step 3.** Log in to the application node and start the RPC version of your application (for instance, `Topaz`), then set `UserName`. For example:

```
Client% topaz
topaz> set username DataCurator
```

- Step 4.** Set `GemNetId` to `gemnetobject`. Because the session process is to run on the server, be sure to include the node name in the `GemNetId` NRS. (It's not necessary to set the `GemStone` login parameter when the Stone repository monitor runs on the same node as the Gem.) For example:

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

- Step 5.** Log in to the repository:

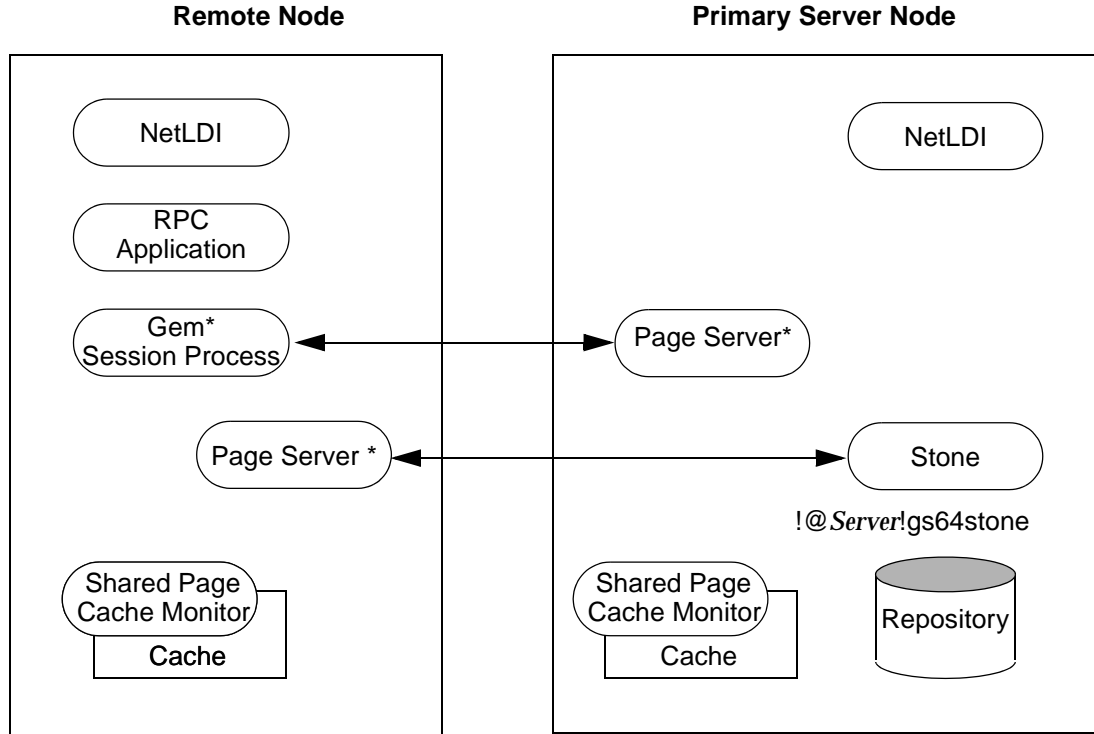
```
topaz> login
GemStone Password?
successful login
topaz 1>
```

At this point, you are logged in to a Gem session process on the server node. That session process acts as a server to `Topaz` RPC and as a client to the Stone.

To Run the Gem and Stone on Different Nodes

The configuration shown in Figure 3.7 is unusual in that the RPC application and its session process are running on the same node. (While this configuration might be desirable during application development, a linked application, if it is available, probably would give better performance.)

The NetLDIs and page servers function similarly to those described for the linked application (see “To Run a Linked Application on a Remote Node” on page 3-20). In Figure 3.7, however, the NetLDI also starts the RPC Gem session process at the request of the application.

Figure 3.7 Starting the Session Process on a Remote Node

Step 1. Unless the NetLDIs are running in guest mode, decide how you will provide access so that application can start a Gem session process on the remote node.

- You can create a `.netrc` file in your home directory on the remote node containing a line like the following, where `userPassword` is your operating system password on the server:

```
machine remoteNode login userName password userPassword
```

- ❑ Alternatively, you can set the application login parameters, such as HostUserName and HostPassword, after you start the application. For example:

```
topaz> set hostusername yourLogin
topaz> set hostpassword yourPassword
```

Step 2. Log in to the remote node and start a NetLDI.

- ❑ 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):

```
Remote% startnetldi
```

- ❑ To start the NetLDI in guest mode (authentication is not required), make sure `$GEMSTONE/sys/netldid` does NOT have the S bit set. Log in as the captive account *name*, then issue this command:

```
Remote% startnetldi -g -aname
```

Step 3. Start the RPC version of your application (for instance, Topaz):

```
Remote% topaz
```

Step 4. Set GemNetId to `gemnetobject`. This network object identifies scripts that start a session process. For example:

```
topaz> set gemnetid gemnetobject
```

Step 5. Set the GemStone name, using NRS syntax to specify its location on the primary server node. Then set the UserName and log in. For example:

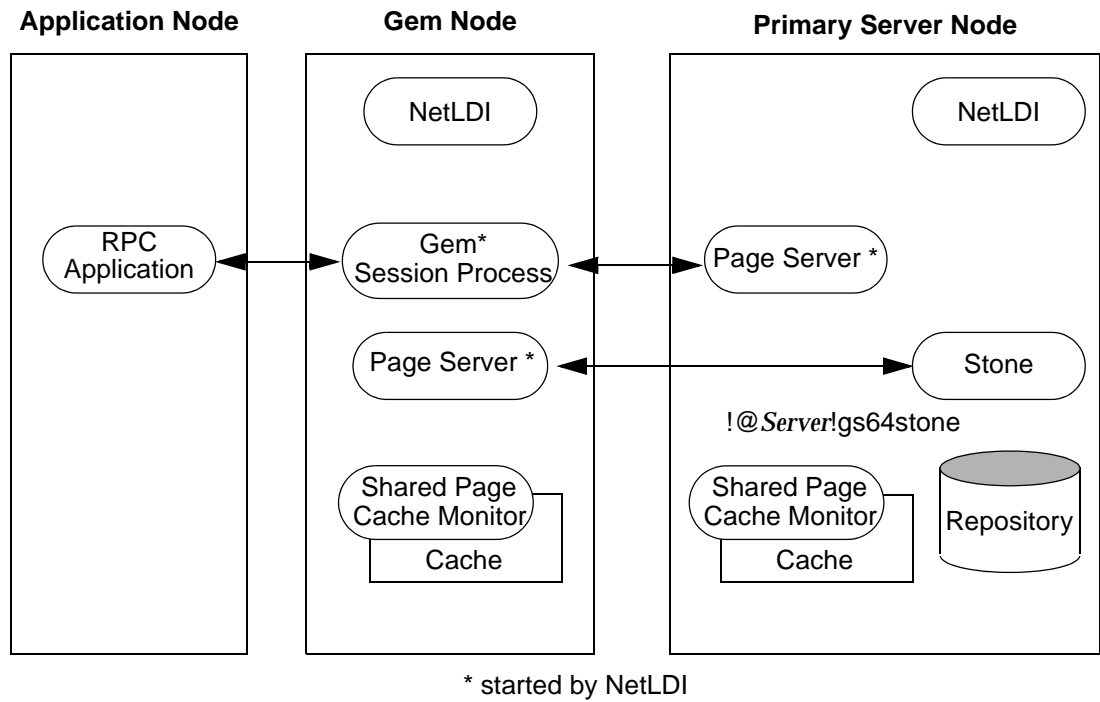
```
topaz> set gemstone !@Server!gs64stone
topaz> set username DataCurator
topaz> login
GemStone Password?
successful login
topaz 1>
```

To Run the Application, Gem, and Stone on Three Nodes

The RPC application, session process, and Stone can run on three separate nodes, as shown in Figure 3.8. The application runs on its node and connects to a Gem session process on the Gem's node. 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 not necessarily on the application node, which runs only the RPC application).

Figure 3.8 Connecting an RPC Application, Three Nodes



The network access problem is similar to that in other RPC configurations: unless the NetLDI on the Gem node is running in guest mode, you must provide authentication to start the Gem session process.

Step 1. Unless the NetLDI on the Gem node is running in guest mode, decide how you will provide authorization for network services on that node.

- ❑ You can create a `.netrc` file in the your home directory on the application node containing a line like the following, where the password is your password on the Gem's node.

```
machine Gem login userLogin password secret2
```

- ❑ You can set the application login parameters, such as `HostUserName` and `HostPassword`, after you start the application. For example:

```
topaz> set hostusername yourLogin
topaz> set hostpassword yourPassword
```

Step 2. Log in to the Gem's node and start the NetLDI.

- ❑ 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):

```
Remote% startnetldi
```

- ❑ To start the NetLDI in guest mode (authentication is not required), make sure `$GEMSTONE/sys/netldid` does NOT have the S bit set. Log in as the captive account *name*, then issue this command:

```
Remote% startnetldi -g -aname
```

Step 3. Log in to the application node. Start the RPC version of your application (for instance, Topaz):

```
Application% topaz
```

Step 4. Set `GemNetId` to `gemnetobject`, and include the location, *gemNode*, in the NRS. For example:

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

Step 5. Use NRS syntax to specify the location and name of the repository. Then set the username and log in. In Topaz, for example, set GemStone and UserName:

```
topaz> set gemstone !@Server!gs64stone
topaz> set username DataCurator
topaz> login
GemStone Password?
successful login
topaz 1>
```

At this point, your Topaz application on the application node has logged you in to a Gem session process on the Gem's node, and the session process has logged in to the repository on the server.

Troubleshooting Remote Logins

Logging in to 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 (see the figure for your configuration). Also make sure that the NetLDIs have the same port number in `/etc/services`. All nodes must be listed in `/etc/hosts`.
- ❑ If an RPC application is being started (that is, one with a separate Gem session process), make sure the user who starts the application has an entry for the Gem's node in a `.netrc` file in `$HOME`, or that other authentication provisions have been taken, such as running the NetLDI in guest mode with a captive account. The owner of the Gem process needs an account on the node where the Gem will run and needs write access to the Gem log, typically in `$HOME`. Ownership and permissions for `$GEMSTONE/sys/netldid` must be appropriate for the authentication system in use (see pages 3-12 and 3-14), and the directories in `/opt/gemstone` must be writable.
- ❑ Make sure that the user who started the Stone has an account on the remote node. This user also must have write permission for `$HOME` so that log files for the remote node can be created, unless steps are taken to create the log files in another directory.

- ❑ Check any GEMSTONE environment variables for definitions that point to a previous version: `env | grep GEM`.

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 Mon Mar 26
16:22:27 PDT 2007
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 "To Set Ownership and Permissions for Session Processes" on page 2-6.

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`. 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 3-32 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 “To Set a Default NRS” on page 3-15.)

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**.

The following example shows how you might use the NetLDI log to diagnose problems during an RPC login.

1. The client (Topaz, GBS, or GCI) contacts the NetLDI and requests a session.

```
Entering Service Loop
0: --- 03/22/07 12:40:11.406 PDT :
    Attempting accept...
        ...succeeded accepting client from 10.80.8.12,
        connection = 2
0: --- 03/22/07 12:40:11.484 PDT :
    Finished reading client request:
        Client is a rpc application.
        '!#encrypted:<username>@password!gemnetobject'
```

If there is no message in the NetLDI log after a login attempt, the failure is in connecting to the NetLDI. (This is usually clear from the error message.) Check your login parameters, particularly the NetLDI name, the port id assigned in the `/etc/services` file, and the `GEMSTONE_NRS_ALL` setting.

If you started NetLDI using the `-p` option to specify a port range, verify that these ports are appropriate and, if you have a firewall, that these ports are open.

2. NetLDI starts listening on a "callback" port. This is not the port that the NetLDI uses to accept connections. This port is just used temporarily for the Gem-NetLDI communication.
3. NetLDI launches `gemnetobject` (or a similar script specified in the login parameters), telling it the number of the callback port it should use.

```
0: --- 03/22/07 12:40:11.499 PDT :
    Successful fork; Child's Pid: 28836 command is:
        '/<$GEMSTONE>/sys/gemnetobject TCP 48273 30'
```

Note that "Successful fork" does not necessarily mean that the Gem was correctly started and initialized; the NetLDI does not block.

4. The `gemnetobject` script runs. If you are not using the default `gemnetobject`, and you have errors in your script, it is very difficult to diagnose them; since the problem is outside of the NetLDI or Gem, details are not reported in either the NetLDI or the Gem logs.

5. The `gemnetobject` script exec's `gem` to start running the Gem's code.
6. The Gem session initializes itself. At this point there is a Gem log. The pid of the `gem` process is shown in the NetLDI log (step 3).
7. The Gem starts listening on a client service port. This is yet another distinct port, and is the one that will be used for ongoing communication between the Gem and the client. The port id is shown in the NetLDI log (step 10).
8. The Gem connects to the NetLDI on the callback port.
9. The Gem tells the NetLDI that it is ready, and the number of its client service port:

```
0: --- 03/22/07 12:40:11.803 PDT :  
    Now reading reply from child
```

If this step does not occur within the NetLDI timeout, the NetLDI times that Gem out and stops listening on the callback port. (The NetLDI log will contain a message to that effect.) If you are getting timeouts with the default timeout interval, try increasing the timeout by invoking `startnetldi` with the `-timeout` option. This may at least help you keep working while determining why the timeout is occurring.

10. NetLDI passes the number of the client service port back to the client (Topaz, GBS or GCI), informing it that a Gem has been started for it, and that this Gem is waiting for a connection at a specific port.

```
0: --- 03/22/07 12:40:11.804 PDT :  
    Reply to client started:  
    'SUCCESS 48274'  
0: --- 03/22/07 12:40:11.804 PDT :  
    Done writing reply to client.  
0: --- 03/22/07 12:40:11.885 PDT :  
    Disposed. elapsed time = 0
```

11. The client contacts the Gem directly on its client service port.

***Part II:
Administering
GemStone/S 64 Bit***

Running GemStone

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

- Starting the GemStone Object Server (page 4-2)
- Starting Network Long Distance Information (NetLDI) servers (page 4-8)
- Identifying running servers (page 4-10)
- Logging in to a GemStone session (page 4-10)
- Identifying the current sessions (page 4-16)
- Shutting down the object server (page 4-17)

Additional topics explain how to:

- Recover from an unexpected shutdown (page 4-19)
- Load objects in bulk (page 4-22)
- Enter and use manual transaction mode, which we recommend that you use as often as possible (page 4-22)

4.1 How to Start the GemStone Server

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

- Where GemStone is installed — The GEMSTONE environment variable must point to the directory where GemStone is installed, such as `/users/g64stone`. 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 A-2.
- 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` as the repository file and places the transaction logs in the same directory. You may want to move these files to other locations. For further information, see “Choosing the Extent Location” on page 1-20.

To Start GemStone

Follow these steps to start GemStone following installation or an orderly shutdown (to recover from an abnormal shutdown, refer to “How to Recover from an Unexpected Shutdown” on page 4-19).

NOTE

In certain distributed installations, a GemStone NetLDI must be running on other nodes before you can start the repository monitor. These situations are discussed in Chapter 3 of this manual and ordinarily do not apply.

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 `GemStone-sparc.Solaris`. For example:

```
$ GEMSTONE=/users/GemStone-sparc.Solaris
$ export GEMSTONE
```

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

- GEMSTONE
- GEMSTONE_SYS_CONF
- GEMSTONE_EXE_CONF
- GEMSTONE_LANG

Step 2. Set your UNIX path. One way to do this is to use one of the `gemsetup` script. 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 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`. For additional information about `startstone`, see page B-15.

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/g64stone.log`), then refer to the discussions that follow this summary.

- The GemStone key file is missing or invalid (see page 4-4).
- The shared page cache cannot be attached (see page 4-4).
- An extent file is missing or cannot be opened for exclusive use because another GemStone process is using it (see page 4-5).
- Because of the timing of a system crash, the repository monitor is trying to create an extent that already exists (see page 4-6).
- A transaction log needed for recovery is missing, or the log directory or device does not exist (see page 4-7).

- The repository has become corrupted (see page 4-7).

The error numbers printed as part of a log message are defined in the file `$GEMSTONE/include/gcierr.ht` and in the *GemStone/S 64 Bit Programming Guide*.

Missing or Invalid Key File

The Stone repository monitor must be able to read the key file `$GEMSTONE/sys/gemstone.key`. Ordinarily, you create this file during installation from information provided by GemStone. Be careful to enter the information correctly, following the instructions on the sheet. If the information is missing, contact GemStone Technical Support as described in the Preface.

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 slightly larger than 10 MB. For specifics about configuring shared memory, refer to your *GemStone/S 64 Bit Installation Guide*.
- 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. Page 1-18 describes a utility (`$GEMSTONE/install/shmem`) that will help you check the configuration.
- 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 "To Set File Permissions for the Server" on page 1-30. 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 (pageKind [null])
```

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) (pageKind [null])
```

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 but before the next checkpoint, 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.
```

Verify 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 not cleanly shutdown, recovery needed  
Repository startup from checkpoint = (fileId 0, blockId 14)  
Searching for most recent transaction log  
no log files found  
Searching for transaction log file, fileId 0, directoryId 0,  
filename = /users/gs64stone/data/tranlog0.dbf  
Error during repository recovery
```

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. If the Stone detects that the logs actually are present, it performs a normal startup. If the log file is present but corrupted, you may have to remove the file before restarting GemStone.

NOTE

*When you use **startstone** with the **-N** option, any transactions occurring after the last checkpoint are permanently lost.*

Repository Failure

If you have GemStone backups (either online extent backups or Smalltalk full backups, as described in Chapter 9), you can restore the repository to the state of the most recent backup. If full logging (STN_TRAN_FULL_LOGGING=True) was in effect at the time of the backup, objects committed by subsequent transactions can then be recovered from the transaction logs. For details, see “How to Restore from an Online Extent Backup” (page 9-7) or “How to Restore from a Smalltalk Full Backup” (page 9-14).

If you do not have a recent backup and transaction logs for valuable data, you may still be able to recover your committed repository. However, this procedure is not nearly as reliable and may be quite time-consuming. See “How to Audit the Repository” on page 8-9.

Other Startup Failures

- Check `/opt/gemstone/locks` and remove old files. On some systems, the lock files may be located in `/usr/gemstone/locks`. 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 you can't start GemStone under any circumstances, try `pageaudit` on the repository. (See "How to Audit the Repository" on page 8-9.) If the page audit is good but GemStone still doesn't start, check your installation configuration. For more help, contact your local GemStone administrator or GemStone Technical Support.

4.2 How to Start a NetLDI

It's common practice to start a GemStone Network Long Distance Information (NetLDI) server when starting a Stone repository monitor. Chapter 3 of this manual describes the two situations in which a NetLDI is necessary:

- A user will be running an RPC application with a separate Gem session process on the Stone's node.
- A user will be running a linked application or a separate Gem on another node and logging in to the repository on the Stone's node.

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 `GemStone64Bit2.2.0-sparc.Solaris`. For example:

```
$ GEMSTONE=/installDir/GemStone64Bit2.2.0-sparc.Solaris
$ export GEMSTONE
```


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):

```
Client% 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:

```
Client% startnetldi -g -aname
```

For additional information about `startnetldi`, see the command description in Appendix B. For information about the authentication modes, see “How to Arrange Network Security” on page 3-9.

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. Inspect 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`.

- 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`.

The error numbers printed as part of a log message are defined in the file `$GEMSTONE/include/gcierr.ht`.

4.3 To List 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` or `/usr/gemstone/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      2.2.0     gsadmin    Mar 18 10:00 cache  gs64stone@nodeA
OK      2.2.0     gsadmin    Mar 18 10:00 Stone gs64stone
OK      2.2.0     gsadmin    Mar 06 12:13 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 NetLDI running.

4.4 How to Start 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 5, “User Accounts and Security.”
- For additional information about logging in from a remote node, see Chapter 3, “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 2-2.

To Define a GemStone Session Environment

In order to start a GemStone session, the following must be defined through your UNIX environment:

- Where GemStone is installed

All GemStone users must have a GEMSTONE environment variable that points to the GemStone installation directory, such as `/installDir/GemStone64Bit2.2.0-hppa.hpux`. The directory `$GEMSTONE/bin` should be in your search path for commands. For an example, see the next topic, “To Start a Linked Session”.

- Which configuration parameters to use

Because each GemStone session can have its own configuration file, some users may need a second environmental variable, such as `GEMSTONE_EXE_CONF`. If no other file is found, the session uses system defaults. For further information, see “To Set Up the User’s Environment” on page 5-17 and “How GemStone Uses Configuration Files” on page A-2.

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 UNIX path for a session are the same as those given on page 4-2 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 `GemStone64Bit2.2.0-sparc.Solaris`. For example:

```
$ GEMSTONE=/installDir/GemStone64Bit2.2.0-sparc.Solaris
$ 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_LANG

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.

```
topaz> login
GemStone Password?
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 Steps 1 and 2 of the previous example.

Step 1. Use **gslist** to find out if a NetLDI is already running. The default name for the NetLDI is *gs64ldi*. (This list also shows the Stone and shared page cache monitor.)

```
% gslist
Status  Version   Owner      Started    Type  Name
-----  -
exists  2.2.0     gsadmin    Mar 18 10:00 cache  gs64stone@nodeA
exists  2.2.0     gsadmin    Mar 18 10:00 Stone  gs64stone
exists  2.2.0     gsadmin    Mar 06 12:13 Netldi  gs64ldi
```

Step 2. If necessary, start a NetLDI:

- 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):

```
Client% 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:

```
Client% startnetldi -g -aname
```

Step 3. Unless the NetLDI is running in guest mode with a captive account, decide how you will provide authentication so that the NetLDI can start the Gem session process. Choose one of the following:

- You can create a `.netrc` file in your home directory containing a line like the following, where *hostName* is the name of this node (which is also the server node):

```
machine hostName login yourUnixId password yourPassword
```

- ❑ You can 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. Start the RPC application (such as `Topaz`), then set the `UserName`.

```
% topaz
topaz> set username DataCurator
```

Step 5. 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 6. Log in to the GemStone session.

```
topaz> login
GemStone Password?
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 `$GEMSTONE/sys/gemstone.key`, but you can examine the comment in that file. For example:

```
# Stone Session limit:      10
```
- The GemStone configuration option `STN_MAX_SESSIONS` (page A-32) 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. By

default, the Stone log file is `$GEMSTONE/data/gemStoneName.log`. Look for a line like this in a box:

```
SESSION LIMIT: Maximum number of concurrent sessions: 64
```

- The GemStone configuration option `SHR_PAGE_CACHE_NUM_PROCS` (page A-22) restricts the number of sessions that can attach to a particular shared page cache. This number can be different on each node, depending on the configuration file that is read by the process that starts the cache. On the node where the Stone runs, one of this number is used by the Stone, the shared page cache monitor, each GcGem (garbage collection) session, each Stone AIO page server, the page manager, the SymbolGem, and each free frame page server. On other nodes, the Stone's page server and the shared page cache monitor each use one. For details, see "To Set the Page Cache Options and the Number of Sessions" on page 1-15. Check the Stone's log for warnings that the value requested for `SHR_PAGE_CACHE_NUM_PROCS` has been adjusted to match your system's configuration.
- The UNIX kernel must provide one semaphore for each session that wants to attach to the shared page cache. See "Reviewing Kernel Tunable Parameters" on page 1-14.
- The UNIX kernel file descriptor limit can restrict the number of sessions, and GemStone executables attempt to raise that limit. For information, see the discussions of "Estimating File Descriptor Needs" on page 1-13 (for the Stone) and page 2-5 (for Gems). On some operating systems, you can examine the kernel limit by invoking `limit`.
- 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 Mar 26
16:21:08 PDT 2007
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 — even on the same node — see "Troubleshooting Remote Logins" on page 3-30.

The error numbers printed as part of a log message are defined in the file `$GEMSTONE/include/gcierr.ht` and in the *GemStone/S 64 Bit Programming Guide*.

4.5 How to Identify Sessions Logged In

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
session number: 6      UserId: DataCurator
```

The session number can be used with other `System` class methods to stop a particular session or to obtain its `UserProfile`. See `stopSession:aSessionId` and `userProfileForSession:aSessionId`.

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.

The method `System class>>descriptionOfSession:aSessionId` returns an array of descriptive information by which you can trace the session name to a particular person: the second element shows the operating system process id (pid),

and the third element shows the name of the node on which it is running. In this example, the DataCurator session is running on “node1” as pid 3010:

```
topaz 1> printit
System descriptionOfSession: 2
%
an Array
  #1 an UserProfile
  #2 3010
  #3 node1
  ...
```

For details about these methods and the information returned, see the class and method comments in the image.

4.6 How to Shut Down the Object Server and NetLDI

Privileges required: SystemAccess and SystemControl.

To shut down GemStone from UNIX, first make sure that all user sessions have logged out. One way to find out about other user sessions is to 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
session number: 6   UserId: DataCurator
```

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 default name during startup was `gs64stone`. If necessary, use **gslis** (page B-7) to find the name.

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). For details about user accounts and privileges, see Chapter 5.

There is a similar command to shut down the NetLDI network service.

```
% stopnetldi [ netLdiName ]
```

For more information about the stopstone and stopnetldi commands, refer to Appendix B, "GemStone Utility Commands."

If you are logged in to a GemStone session, you can invoke `System class>>shutDown`, which also requires the SystemControl privilege.

CAUTION

*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 can perform an orderly shutdown.*

*Do NOT use **kill -9** or another uncatchable signal, which may not result in a clean shutdown or may cause the Stone repository monitor to shut down when you intended to kill only a Gem process. If for some reason you 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.*

4.7 How to Recover 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. Unless you specified another file on the **startstone** command line, the Stone log is

`$(GEMSTONE)/data/gemStoneName.log`.

The `$(GEMSTONE)/data` directory also contains log files for the Stone child processes. The child processes have log names formed from *gemStoneName*, the process id, and a descriptive abbreviation. For instance:

<code>gs64stone.log</code>	Stone repository monitor
<code>gs64stone_14033admingcgem.log</code>	Admin GcGem
<code>gs64stone_2963pcmon.log</code>	Shared page cache monitor
<code>gs64stone_2967pgsvrff.log</code>	Free Frame page server
<code>gs64stone_2984pgsvraio.log</code>	AIO page server
<code>gs64stone_2987pagemanager.log</code>	Page Manager
<code>gs64stone_2992reclaimcgem0-6.log</code>	Reclaim GcGem
<code>gs64stone_2994symbolgem.log</code>	SymbolGem

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.

Normal 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:

```
--- 03/01/07 08:59:58 PST ---  
'LoginsSuspended' is now set to 1 by DataCurator.  
GemStone is stopping the Admin Gem with sessionId 3,  
processId 25909  
Stopping Reclaim Gem for extents 0 to 1, sessionId 2  
processId 25908  
GemStone is stopping the Symbol Creation Gem with  
sessionId 4, processId 25910  
SHUTDOWN command was received from user DataCurator.  
Now stopping GemStone.
```

After a normal shutdown, restart GemStone in the usual manner. For instructions, see “How to Start the GemStone Server” on page 4-2 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/g64stone/data/extent0.dbf  
PageId = 94  
File = /users/g64stone/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 to tape 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, the procedure you need to follow depends on what was done at the operating system level. For a discussion of the options, see the section “How to Recover After Repair of the File System” on page 9-37.

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_pcmonnnnn.log`, may indicate the reason.

```
--- 03/31/07 15:07:19 PDT ---
```

```
The stone's connection to the local shared cache monitor  
was lost.
```

```
Error Text: 'Network partner has disconnected.'
```

Check `/opt/gemstone/locks` or `/usr/gemstone/locks` and remove any entries left by the monitor that died. These files have names that include the Stone name and a network address, such as `gs64stone@127.0.0.1` and `gs64stone@127.0.0.1..LCK` for the default Stone name `gs64stone`.

The unexpected shutdown of a Gem process may 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, and each Gem must release any locks it holds in the process of shutting down. This error may result from a system 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 “How to Start the GemStone Server” on page 4-2.

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_ERROR` option (page A-29). When that option is set to True (the default) 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
```

You can change this response by setting the `STN_HALT_ON_FATAL_ERROR` configuration option 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.

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 UNIX 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 “How to Audit the Repository” on page 8-9).

If the page audit fails, read the advice under “Disk Failure or File System Corruption” on page 4-20 of this chapter, 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 “How to Start the GemStone Server” on page 4-2.
3. If the restart fails, you may have to restore the repository. For details, see the restore procedures in Chapter 9.

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 on page 4-2. For information about the **startstone** command, see page B-15.

4.8 How to Bulk-Load Objects

During bulk loading of objects into the repository, it may be desirable to make the following changes:

- You may need to commit incrementally, but if so, commit as seldom as possible. There is a limit on how large a transaction can be, either in terms of the total size of previously committed objects that are modified, or of the total size of temporary objects that are transitively reachable from modified committed objects.

To address this concern, increase the `GEM_TEMPOBJ_CACHE_SIZE` configuration option. The size of each transaction (the number of 16 KB pages

written) should be approximately 1/3 to 1/2 the size of GEM_TEMPOBJ_CACHE_SIZE, and no more than 1/4 to 1/2 the size of the shared page cache.

- If you are loading through GemBuilder, you can reduce growth of your Smalltalk image by using forwarders or explicit stubbing. For instance, when adding objects to a large collection, make the Collection object a forwarder or, after adding each element, send it the message `#stubYourself`.
- If the bulk load consists of large transactions, put the repository in partial logging mode during loading (`STN_TRAN_FULL_LOGGING = False`) and lower the `STN_TRANLOG_LIMIT` configuration option. This change reduces size of the resulting transaction logs by causing each transaction larger than the specified limit to be written as a checkpoint. (The `STN_TRANLOG_LIMIT` configuration option has no effect if the repository has been run in full logging mode.)
- Alternatively, you can increase performance during bulk loads by adding the following entries to your configuration file:

```
STN_TRAN_LOG_DIRECTORIES = /dev/null, /dev/null;  
STN_TRAN_FULL_LOGGING = TRUE;
```

For information about these configuration file options, see pages A-36 and A-35, respectively.

NOTE

Be aware that using `/dev/null` for the tranlog directories will prevent you from being able to restore tranlogs in the event of a system failure.

4.9 Considerations for Large Repositories

GemStone/S 64 allows you to define a very large shared page cache, thereby enabling you to run very large repositories. This section presents special considerations that apply to large repositories.

Loading the object table at startup

When starting the repository, the object table is not loaded into memory, and initial accesses can take an excessively long time. If you encounter this condition, you may choose to run the **startcachewarmer** utility, which explicitly loads the object table into memory. There is an initial cost at startup, but subsequent performance will be acceptable.

For details about **startcachewarmer**, see page B-11.

Making efficient use of a large cache

When running a system on which many users log in simultaneously, consider using remote caches so that you don't need to run all Gem processes on the same machine. There are a couple of ways to optimize this. The following configuration options are of particular interest:

- To allow Gems to make more efficient use of the large cache, set the `GEM_PGSRV_FREE_FRAME_CACHE_SIZE` configuration option (page A-16) to increase the size of the Gem free frame cache. For example:

```
GEM_PGSRV_FREE_FRAME_CACHE_SIZE = 25;
```

- To improve performance on remote caches, set `GEM_PGSRV_UPDATE_CACHE_ON_READ` (page A-17) to `True` so that remote Gem sessions will update their local caches. For example:

```
GEM_PGSRV_UPDATE_CACHE_ON_READ = TRUE;
```

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 sending `#rtErrSignalAbort` (also called "sigAbort"). If the session does not abort, the Stone repository monitor reinitializes the session or terminates it, depending on the value of `STN_GEM_LOSTOT_TIMEOUT`.

Sessions that are in transaction are immune from this process. 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 *GemStone/S 64 Bit 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 4.1 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 #rtErrSignalFinishTransaction. 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.

For more information on these signals, see the *Programming Guide* for GemStone/S 64 Bit.

User Accounts and Security

This chapter tells how to use the GemStone/S 64 Bit tools to perform administrative tasks on an object server. There are two such tools:

- The Topaz programming environment is a line-oriented interface. It is part of the GemStone Object Server distribution.
- GemBuilder for Smalltalk is available as a separate product and requires Smalltalk from a third-party vendor.

The Topaz interface is available to all administrators and can be used from any terminal and readily accepts input from scripts. All administrative tasks can be performed from the Topaz interface, and a few tasks, such as restoring a backup, require it.

GemBuilder for Smalltalk (GBS) requires a window system. All administrators with access to GemBuilder can use its visual tools for creating and maintaining user accounts. Administrators who are experienced with this interface may also prefer to use its workspace for other administrative tasks, in which case the Topaz examples in this manual should be helpful because the Smalltalk code is the same.

This chapter also shows you how to perform some common GemStone user administration tasks:

- How to create and modify user accounts, including passwords, privileges, group memberships, and symbol resolution, and how to control the user's read-write access to objects through the use of *segments*.
 - Procedures for performing these tasks with the GemBuilder administration tools begin on page 5-11.
 - Procedures for performing these tasks with Topaz begin on page 5-27.
- How to configure GemStone login security by restricting valid passwords, imposing password and account age limits, and monitoring intrusion attempts.

To perform any of the tasks described in this chapter, you should either have the GemBuilder administration tools available or be familiar with the Smalltalk programming language as described in the *Programming Guide for GemStone/S 64 Bit*.

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 segment. This chapter introduces these concepts. For a full description, see the chapter of the *Programming Guide for GemStone/S 64 Bit* that discusses security.

5.1 The Administrative Accounts

For system administrative work, you will use two logins: DataCurator and SystemUser. The DataCurator account is used to perform system administration tasks. The SystemUser account ordinarily is used only for performing GemStone system upgrades. To log in as SystemUser, simply substitute that name for DataCurator when you set the GemStone user name. Access to both of these accounts should be restricted.

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.*

Two other administrative accounts, GcUser and SymbolUser, are for the special sessions that log in to perform the garbage collection and symbol creation tasks, respectively.

- Because GcUser is logged in automatically by the Stone repository monitor, the primary reason to log in as GcUser yourself is to tune garbage collection parameters that are stored in GcUser's UserGlobals.
- Under normal circumstances, you should never need to log in as SymbolUser.

5.2 Defining Your GemStone Environment

Before you can launch one of the administrative tools, it's necessary to define the session environment in UNIX. That process is the same as the one described for all users on page 4-11. Define the GEMSTONE environment variable and be sure that `$GEMSTONE/bin` is in your path.

5.3 User Accounts

This section provides background information about how GemStone stores user accounts, what accounts are predefined, and what determines an account's name space.

UserProfiles

Each GemStone user is associated with an instance of class UserProfile. That UserProfile object contains information describing objects that the user is allowed to examine or modify, messages that the user is permitted to send, the user's native language, and default attributes of any objects that the user creates.

The following paragraphs describe each of the elements that you specify when creating a new UserProfile. If you have the necessary privileges, you can also modify these elements. In addition, the UserProfile contains a symbol list for use in resolving symbols in that user's name space. For a discussion, see "The UserProfile and Session Symbol Lists" on page 5-8.

User ID	Each UserProfile is associated with a <code>userId</code> —a unique String that identifies the user to the GemStone system at login. Embedded spaces are permitted.
Password	The user supplies this password (an <code>InvariantString</code>) for identification purposes at login. This password has no

connection with a user's operating system password and should be different. GemStone stores the password in encrypted form in a secure manner. Users must have explicit privilege to change their own passwords—or anyone else's. (See the discussion of privileges below.)

GemStone provides a number of ways to restrict the passwords that a user can choose, and it can record login failures and disable the account if failed attempts persist. For information about changing the default settings, see "How to Configure GemStone Login Security" on page 5-42.

Default Segment

When you add a new user to the GemStone system, you can either allow the default segment to be nil, or specify a segment to determine the default read and write authorizations for objects created by the user.

In GemStone Smalltalk, a segment groups objects for purposes of authorization (protection); that is, if you can read or write one of a segment's objects, you can read or write all of them. The owner of a segment can designate named groups whose members are authorized to read or write objects in that segment. For more information about segments, see the chapter in the *Programming Guide for GemStone/S 64 Bit* that discusses security.

A default segment 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.

To specify a default segment for a user, you may use an existing segment, or you must create and commit the segment before it can be used.

Privileges

When you create a new UserProfile, you determine whether the new user may perform certain "privileged" system functions that are customarily performed by you, as the GemStone data curator. For example, many of the messages to System require explicit privilege. For developers, you must also specifically grant the privilege that allows the user to modify code. Table 5.1 lists the Smalltalk methods associated with each GemStone privilege.

Note that privileges are more powerful than segment authorization (discussed above). Although the owner of a segment can always use authorization protocol to restrict read or write access to objects in a segment, you (as the data curator) can override that protection by sending privileged messages that let you change the authorization scheme.

Groups

GemStone uses group membership to supervise access to objects; each user can examine or modify only those objects which you (or the segment's owner) have made available to that user, or objects which do not have a segment. Similarly, GemStone ensures that other users cannot see or change objects that you and the owner have agreed to keep private.

Each GemStone user may belong to any number of groups. There are three predefined groups: System, Publishers, and Subscribers. By default, all new users become members of group Subscribers.

Table 5.1 Smalltalk Methods with GemStone Privileges

Type of Privilege	Privileged Methods (some methods require more than one privilege)
SystemControl	GsSession >>stop Repository >>auditWithLimit:reclaimAll:, quickObjectAuditWithLevel:, reclaimAll System class>>changeCacheSlotFreeFrameLimit:to:, changeCacheSlotIoLimit:to:, flushAllExtents, resumeCheckpoints, resumeLogins, sendSigAbortToSession:, shutDown, startCheckpointAsync, startCheckpointSync, stopUserSessions, suspendCheckpointsForMinutes:, suspendLogins
SessionAccess	GsSession class>>sessionWithSerialNumber:, serialOfSession:, sessionIdOfSerial: System class>>currentSessionNames, descriptionOfSessionSerialNum:, descriptionOfSession:, otherSessionNames, stopUserSessions, userProfileForSession:
UserPassword	UserProfile >>oldPassword:newPassword:
DefaultSegment	UserProfile >>defaultSegment:
CodeModification	<i>[You must have CodeModification privilege to create or modify instances of GsMethod, GsMethodDictionary, or Class. Also see the discussion following this table.]</i>

Table 5.1 Smalltalk Methods with GemStone Privileges (Continued)

Type of Privilege	Privileged Methods (some methods require more than one privilege)
OtherPassword	<p><i>[You must have OtherPassword privilege to make any changes to a UserProfile or to the AllUsers UserProfileSet. This includes adding or removing a SymbolDictionary to/from a SymbolList that is not your own.]</i></p> <p>UserProfile>>activeUserIdLimit, activeUserIdLimit:, clearOldPasswords, isDisabled, lastLoginTime, lastPasswordChange, loginsAllowedBeforeExpiration, loginsAllowedBeforeExpiration:, password:, reasonForDisabledAccount, reasonForDisabledAccount:, userId:</p> <p>UserProfileSet>>findDisabledUsers, findProfilesWithAgingPassword</p>
SegmentCreation	Segment class>>newInRepository:
SegmentProtection	Segment >>group:authorization:, ownerAuthorization:, worldAuthorization:
FileControl	<p>Repository>>abortRestore, addTransactionLog:size:, commitRestore, continueFullBackupTo:MBytes:, createExtent:, createExtent: withMaxSize:, fullBackupTo:MBytes:, restoreFromArchiveLogs, restoreFromBackup:, restoreFromBackups:, restoreFromCurrentLogs, restoreToEndOfLog:, restoreStatus, restoreStatusNextFileId, restoreStatusOldestFileId, startNewLog, timeToRestoreTo:</p>
GarbageCollection	<p>Repository>>auditWithLimit:reclaimAll, findDisconnectedObjectsAndWriteToFile:pageBufferSize:saveToRepository:, markForCollection, markGcCandidatesFromFile:, markGcCandidatesFromFile:forceOnError:, objectAuditNoReclaim, pagesWithPercentFree:, postReclaimAll:, quickObjectAuditWithLevel:, reclaimAll, repairWithLimit:</p> <p>System class>> startAdminGcSession, startAllGcGemSessions, startAllReclaimGcSessions, startReclaimGemForExtentRange:to:, startReclaimGemForExtentRange:to:onHost:, startReclaimGemForExtentRange:to:onHost:stoneHost: startSymbolCreationSession, stopAdminGcSession, stopAllGcSessions, stopAllReclaimGcSessions</p>
(various)	System class>>configurationAt:put:

For a more general discussion of UserProfiles in GemStone, see the discussion of sessions and UserProfiles in the *Programming Guide for GemStone/S 64 Bit*. Also see the instance protocol for UserProfile and UserProfileSet in the image.

Code Modification

CodeModification privilege is required to execute any method that modifies Smalltalk code:

- You must have CodeModification privilege to create instances of GsMethod, or to create or modify instances of GsMethodDictionary or Class. (You cannot modify a GsMethod once it has been created.)
- You must have CodeModification privilege to add a Class to, or remove a Class from, a SymbolDictionary and its subclasses.
- You must have CodeModification privilege to add or remove a SymbolDictionary from your own SymbolList. (See pages 5-21, 5-34, and 5-35.)
- You cannot use GemBuilder for C to modify instances of the following classes (or their subclasses): GsMethod, GsMethodDictionary, Class, SymbolDictionary, SymbolList, UserProfile.

Predefined Users

When GemStone is first installed, the AllUsers object (a UserProfileSet) has UserProfiles already defined for the following users. *You must never delete these users.*

SystemUser	The SystemUser account is the owner of the SystemSegment, which contains the kernel classes. This account ordinarily used only to perform GemStone system upgrades. DO NOT use this account for ordinary administration tasks. For more about this, see “The SystemUser Account” on page 5-8.
DataCurator	The DataCurator account is the account you should use for day-to-day administration tasks. Initially, DataCurator is granted all privileges and belongs to all predefined groups All GemStone UserProfiles are part of the DataCurator Segment.
GcUser	The GcUser account is a special account that logs in to the repository to perform garbage collection tasks. Initially, GcUser has only the GarbageCollection privilege and belongs only to group Subscribers.
SymbolUser	The SymbolUser account is a special account that is used to perform symbol creation tasks. Initially, SymbolUser has only the GarbageCollection privilege and belongs only to group Subscribers . Under normal circumstances, you should never need to log in as SymbolUser.

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.

For more information about AllUsers and other predefined system objects, see Appendix D, “GemStone Kernel Objects.”

The SystemUser Account

SystemUser is a special user, analogous to `root` in UNIX. SystemUser has all privileges, belongs to all predefined groups, and is authorized to read and write all segments. These privileges cannot be taken away, so even if SystemUser is not a member of a specific group, SystemUsers is still able to write to that group's segment, no matter what permissions are set on that segment. SystemUser is the only user in a GemStone system that “breaks the rules” in this way. All other users have only the privileges that you give them.

SystemUser should **only** be used when performing administration tasks that cannot be performed by DataCurator. Normally, the only tasks that require SystemUser login are installation or upgrade of GemStone products.

If you accidentally create objects in SystemSegment, you can move them into their proper segment by sending each object the message `#changeToSegment :`. To do this, you will need to log in as SystemUser.

You cannot disable the SystemUser and DataCurator accounts. Also, their sessions do not expire.

The UserProfile and Session Symbol Lists

As explained in the *Programming Guide for GemStone/S 64 Bit*, the GemStone Smalltalk compiler follows a well-defined path in looking for the objects named by source code symbols (variable names). 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. (The predefined SymbolDictionaries are described later.)

For each session, a persistent instance of class SymbolList is stored in the repository and is referenced from the UserProfile associated with this session 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 and the `GsCurrentSession` object, refer to the *Programming Guide for GemStone/S 64 Bit*.

The UserGlobals SymbolDictionary

When you set up a new `UserProfile`, GemStone automatically creates a new `SymbolDictionary` for the user's private symbols and inserts it as the first element in the symbol list. This new *UserGlobals* `SymbolDictionary` initially contains the following keys:

- `#UserGlobals`—the `UserGlobals` dictionary itself (as the value).
- `#NativeLanguage` —The user's native language, in which GemStone will deliver error messages and dates. Of course, the necessary dictionaries in that language must be created and installed for this to happen. When you add a new GemStone user, the initial `#NativeLanguage` value is `#English`. (For more information, see the discussion of error handling in the *Programming Guide for GemStone/S 64 Bit*.)

The `UserGlobals` dictionary will also contain entries that define the user's private objects (for example, test data and new classes that will not be shared with other GemStone users).

The Globals SymbolDictionary

The second element in each user's initial symbol list is a "system globals" `SymbolDictionary`, ordinarily called *Globals*. This dictionary contains all of the GemStone Smalltalk kernel classes (`Object`, `Class`, `Collection`, and so forth). Although users can read the objects in `Globals`, ordinarily they cannot modify objects in that Dictionary. For more information about the `Globals` Dictionary, see Appendix D, "GemStone Kernel Objects."

The Published SymbolDictionary

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.

Sharing Objects

As described above, the Globals dictionary provides all GemStone users with access to such objects as the kernel classes Integer and Collection. If you want GemStone users to share other objects as well, you need to arrange for references to those objects to be added to the users' symbol lists. 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. If you are using the GemBuilder administration tools, the procedure is similar to that described on page 5-21 for copying a dictionary, except that you select an object in the **Entries** pane and choose **Edit > Copy Entry**. If you are using Topaz, send the message `Published at: aKey put: aValue`.
- You can have users add a special dictionary, such as an application dictionary, to their own symbol list. The procedure is described under "To Add a Dictionary to a Symbol List" on pages 5-21 and 5-35.
- 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 GemStone/S 64 Bit*.

Make sure that the SymbolDictionaries in each user's symbol list include the names of all objects the user might need. For example, you might add each member of a programming team to group Publishers. After completing the definition of a new class, a programmer could make the class available to colleagues by adding it to the Published dictionary.

For more information, refer to the chapter on symbol resolution and object sharing in the *Programming Guide for GemStone/S 64 Bit*.

5.4 Using GemBuilder for Administration

Performing GemStone system administration from GemBuilder for Smalltalk involves these steps:

1. Start GemBuilder and open the GemStone Session Browser. Log in as described below.

To start GemBuilder, it must have been properly installed in your Smalltalk image according to the instructions in your *GemBuilder Installation Guide*.

2. In the **GemStone** menu, choose one of the GemBuilder browsers or tools from the **Admin** menu.
3. Using the tools, administer user accounts or assign segment authorizations for security purposes.
4. When you finish, remember to commit your transaction so that it becomes a permanent part of the repository.

The following sections describe this series of steps more fully.

Logging in Through GemBuilder

1. To open a GemStone Session Browser, choose **Tools > Session Browser** from the **GemStone** menu.

NOTE

The appearance of your GemStone Session Browser may vary somewhat from the illustrations shown here.

2. In the Session Browser (Figure 5.1), click **Add**.
3. When the GemStone Login Editor appears (Figure 5.2), fill in the session parameters. For example:

GemStone repository	gs64stone
GemStone username	DataCurator
GemStone password	(type the DataCurator password)

Then click **OK**.

4. To log in, select the name of the session in the upper left pane of the Session Browser (Figure 5.1). Then click **Login Lnk** or **Login Rpc**.

For complete information about GemBuilder, see the GemBuilder for Smalltalk manual.

Figure 5.1 The GemStone Session Browser

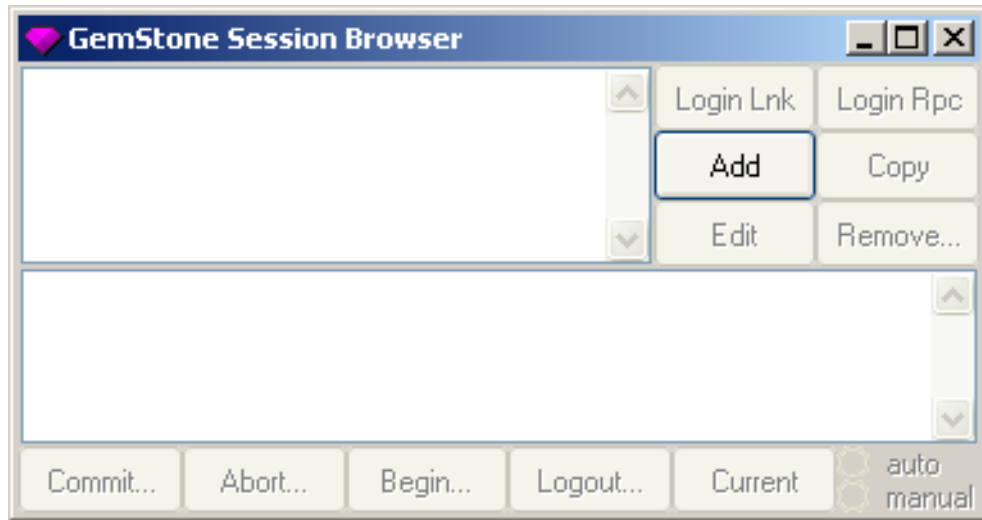


Figure 5.2 The Login Editor

GemStone Login Editor

GemStone repository: gs64stone

GemStone username: [Empty]

GemStone password: [Empty]

Remember

Host username: [Empty]

Host password: [Empty]

Remember

Gem Service: gemnetobject

OK Cancel

Finding the GemBuilder Administration Tools

Once you have successfully logged in to GemStone, you can select any of the GemBuilder browsers or tools from the **GemStone** menu and begin working with the repository.

Figure 5.3 (page 5-15) shows the tools that are available through the **Admin** menu:

- The **Users** tool lets you examine, modify, and delete existing accounts or create new ones. You select a user account in the GemStone User List dialog. The GemStone User and Privileges dialogs show the UserProfile for the selected account.
- **Symbol Lists** lets you examine and modify the name space of a particular user. You can use the SymbolList Browser to add and delete dictionaries from the user's SymbolList, as well as examine and modify the entries in the dictionaries that make up the SymbolList.
- **Segments** lets you create segments and set their authorizations. Segments provide the means for managing GemStone authorization at the object level by assigning objects to segments that have appropriate authorization characteristics.

Committing Your Changes

Remember to commit your changes to the repository. To commit your changes, do one of the following:

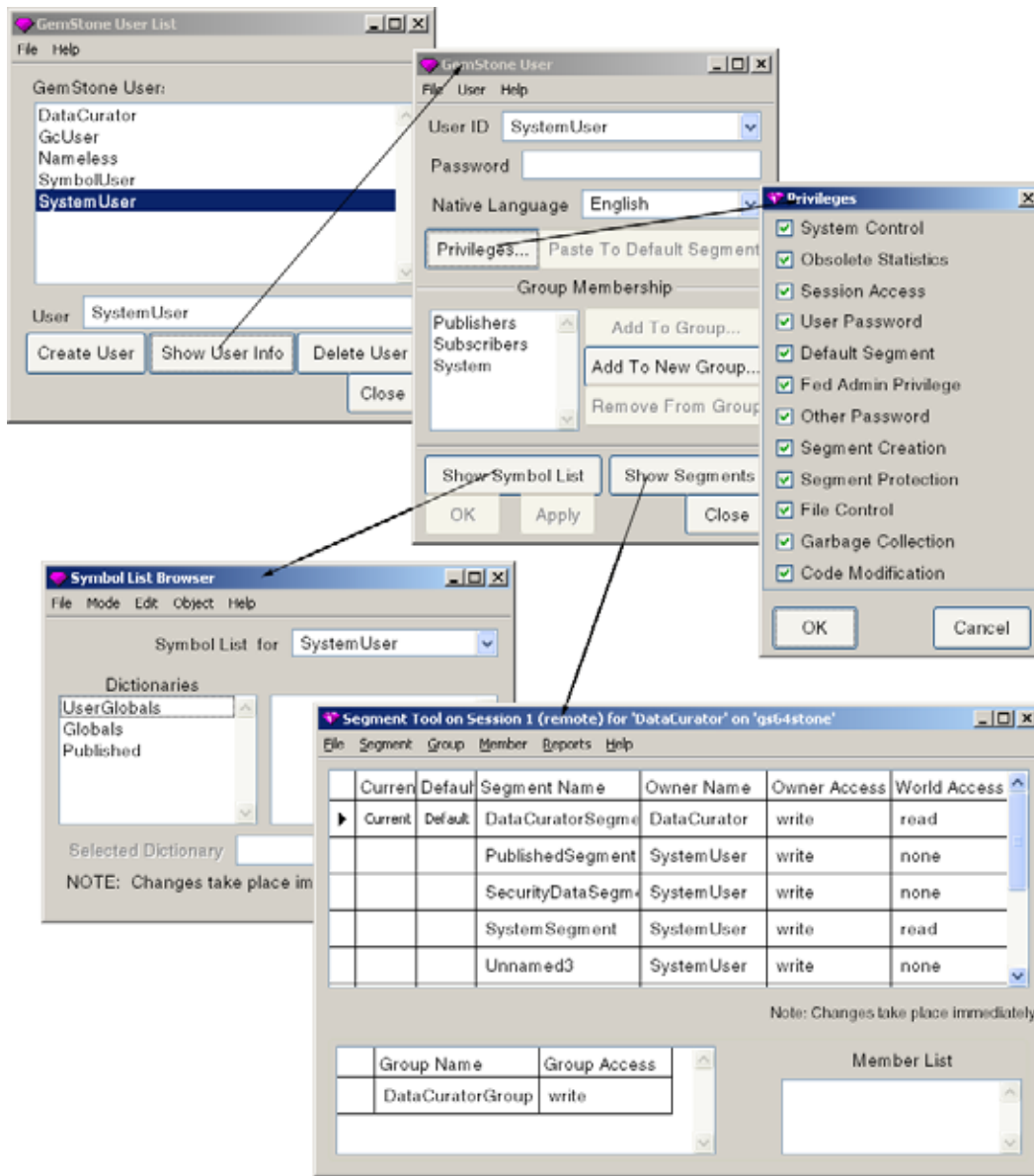
- Click the **Commit** button in the Session Browser (Figure 5.1 on page 5-12).
- Choose **Commit** from the **File** menu of the administration tools, as shown in Figure 5.3.

If you log out after performing work and do not commit it to the permanent repository, the uncommitted work is lost.

Logging Out

To log out of GemStone from the Session Browser (Figure 5.1), select your session in the browser's lower pane, then click **Logout**. When you log out, GemBuilder prompts you to commit your changes.

Figure 5.3 The GemBuilder Administration Tools



Administering User Accounts

You can use the GemBuilder administration tools to create user accounts (UserProfiles) and examine, modify, and remove existing accounts. Most of these tasks use the GemStone User List (left side of Figure 5.4) to open a GemStone User dialog for a particular user (right side of Figure 5.4).

Follow these general steps for most tasks described in this section:

Step 1. In the **GemStone** menu, choose **Admin > Users**.

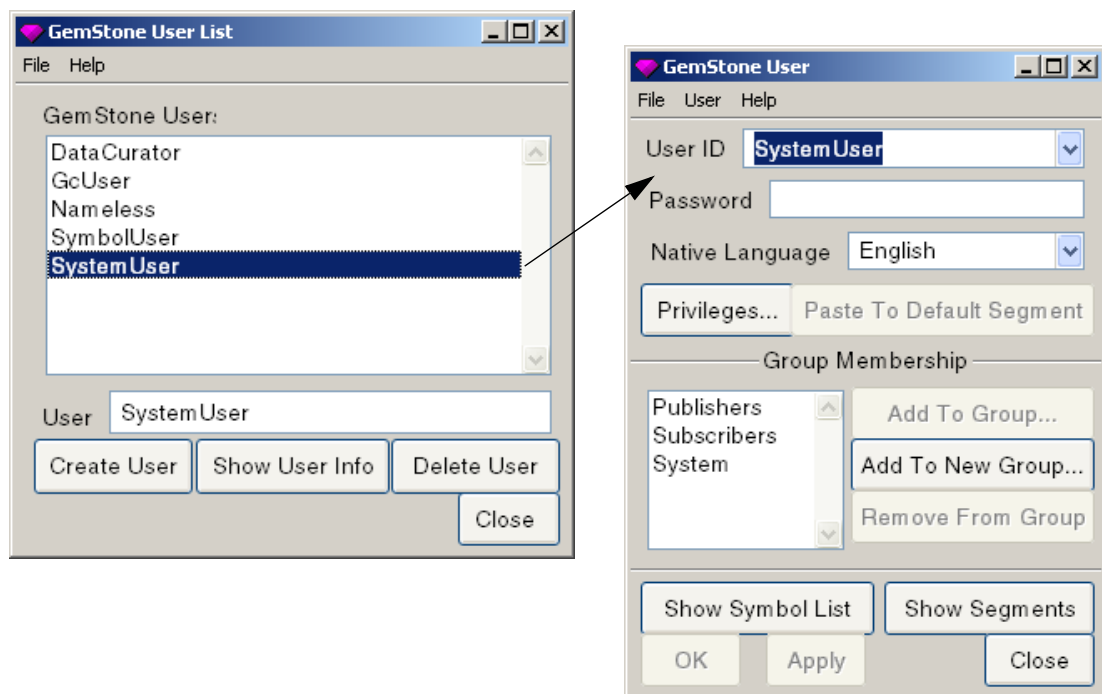
Step 2. When the user list appears, select the UserId, then click **Show User Info**.

Step 3. When the GemStone User dialog appears, carry out the steps in the specific procedures that follow.

Step 4. When you are finished, click **OK** or **Apply**.

Step 5. Remember to commit your transaction before logging out.

Figure 5.4 GemBuilder Tools for Accessing User Profiles



To List Existing Users

Privileges required: None.

In the **GemStone** menu, choose **Admin > Users**. The resulting GemStone User dialog lists all users defined in the repository.

For a list of users that includes each user's group memberships, choose **Admin > Segments** from the **GemStone** menu. When the Segment Tool appears, choose **User Report** from the **Reports** menu.

To Add a User

Privileges required: #OtherPassword privilege and write authorization to the segment of AllUsers.

When you add a user, the GemStone User dialog creates a new UserProfile object and stores it with other UserProfiles in the global object AllUsers (a UserProfileSet). By default, the new UserProfile has no segment, no privileges and no group membership.

Step 1. In the **GemStone** menu, choose **Admin > Users**.

Step 2. When the GemStone User List (Figure 5.4 on page 5-16) appears, click the **Create User** button.

Step 3. When an empty GemStone User dialog appears, enter the **User ID**.

Step 4. If you enter a **Password**, it will not be echoed, but you will be asked to verify it. If you do not enter a password, it will be set to "gemstone". The password must not be the same as the UserId and must not be longer than 1024 characters.

Step 5. You can also set privileges and group membership at this time. These operations are described on page 5-20 and page 5-23, respectively.

Step 6. Click **OK** when you are finished, or click **Apply** if you want to add another user.

Step 7. Choose **Commit** from the **File** menu.

Step 8. Set up the user's environment, as explained next.

To Set Up the User's Environment

In addition to adding the UserProfile, you should make sure that each new user has the GEMSTONE environment variable defined so that it points to the

installation directory. Each user also must know how to execute `$GEMSTONE/bin/gemsetup` as described under “To Start a Linked Session” on page 4-11, or must be able to perform similar functions within an initialization file.

Users may need to belong to the same UNIX group as the Stone repository monitor process so that they can access the GemStone extents and shared page cache. For further information, see “To Set File Permissions for the Server” on page 1-30.

Depending on your site requirements and the applications that they are running, users may need to set one or more of the following environment variables:

- `GEMSTONE_SYS_CONF` and `GEMSTONE_EXE_CONF`, which can point to customized configuration files for specific GemStone servers and user sessions. For general information, see “Creating or Using a System Configuration File” on page A-6.
- `GEMSTONE_NRS_ALL`, which can set a number of network-related defaults, including the type of user authentication that GemStone expects. For further information, see “To Set a Default NRS” on page 3-15.
- `GEMSTONE_LOG`, which can set the location of system log files for the Stone repository monitor and its child processes. For further information, see “GemStone Server Logs” on page 8-2.
- `GEMBUILDER`, which points to the directory where GemBuilder is installed. See the documentation for your version of GemBuilder for details about the need for this or other environment variables.

Users may require write access to their home directory in order to create `.netrc` files and certain log files. At sites where such access is not permitted, refer to “How to Arrange Network Security” on page 3-9 and to the discussion of `GEMSTONE_NRS_ALL` on page 3-15.

To Remove a User

Privileges required: OtherPassword and write authorization in the segment of AllUsers.

When you delete a user's account, the Users tool removes the UserProfile from AllUsers and puts the UserProfile in your UserGlobals as *oldUserID_userProfile* so that objects owned by the former user can still be accessed. That user's persistent SymbolList is saved in your UserGlobals as *oldUserID_symbolList*. If you remove the UserProfile and SymbolList, the objects may be discarded.

Step 1. In the **GemStone** menu, choose **Admin > Users**.

Step 2. When the GemStone User List (Figure 5.4 on page 5-16) appears, select the UserId.

Step 3. Click **Delete User**. You will be asked to confirm the action. If you proceed, the old UserProfile and SymbolList will be saved in your UserGlobals.

Step 4. Remember to commit your transaction before logging out.

To Change a Password

Privileges required: `UserPassword` to change your own password; `OtherPassword` to change another user's password.

The new password will take effect on the next login after you commit the current transaction.

Your choice of passwords for your own account may be subject to optional constraints as to pattern and the use of certain words. For information, ask your system administrator or see "How to Configure GemStone Login Security" on page 5-42.

Step 1. In the **GemStone** menu, choose **Admin > Users**.

Step 2. When the GemStone User List (Figure 5.4 on page 5-16) appears, select the `UserId`, then click **Show User Info**.

Step 3. Enter the new password in the GemStone User dialog, then press `Tab` or `Return`. The password must not be the same as the `UserId` and must not be longer than 1024 characters.

Step 4. When the verification window appears, enter the new password again.

Step 5. Choose **OK** or **Apply**. If you are changing your own password, you will be asked to enter the old password.

Step 6. Remember to commit your transaction before logging out.

To Change a User's Privileges

Privileges required: write authorization to the segment of the user's `UserProfile`; `OtherPassword`.

The new privileges will take effect when you commit the current transaction.

Step 1. In the **GemStone** menu, choose **Admin > Users**.

Step 2. When the GemStone User List (Figure 5.4 on page 5-16) appears, select the `UserId`, then click **Show User Info**.

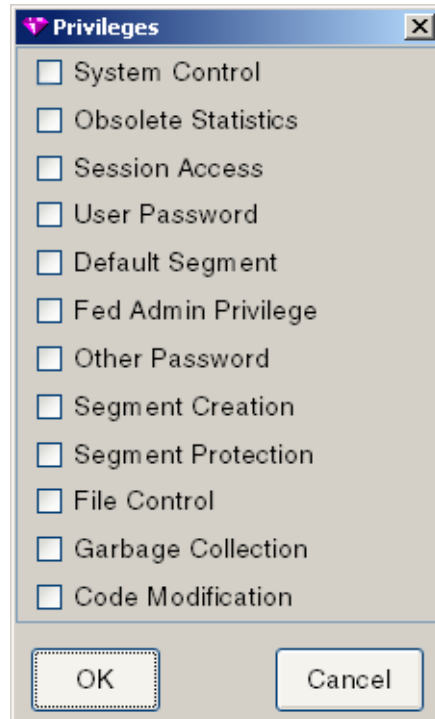
Step 3. In the GemStone User dialog, click the **Privileges** button.

Step 4. When the Privileges dialog (Figure 5.5) appears, click on the check boxes to toggle each privilege that you want to change.

Step 5. Click **OK** when you are finished.

Step 6. Remember to commit your transaction before logging out.

Figure 5.5 The Privileges Dialog



To Add a Dictionary to a Symbol List

Privileges required: To add a SymbolDictionary to your own symbol list, CodeModification. To add a SymbolDictionary to another user's symbol list, OtherPassword privilege and write permission to the segment of the other user's SymbolDictionary.

You can copy a dictionary from one symbol list and add it to a different user's persistent symbol list by using the Symbol List Browser. 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.

Step 1. In the **GemStone** menu, choose **Admin > Symbol Lists**.

Step 2. When the Symbol List Browser appears, open **Symbol List for** and select a **userId** that already has the dictionary in its symbol list. See Figure 5.6.

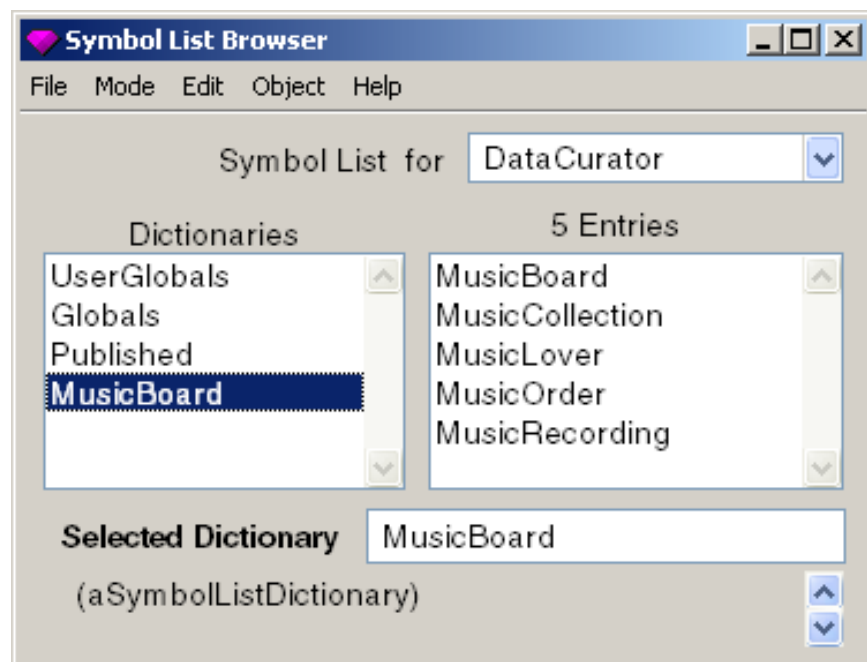
Step 3. In the **Dictionaries** pane, select the dictionary you want to copy. Then choose **Edit > Copy Dict**.

Step 4. Open **Symbol List for** again. This time, choose the **userId** where you want to add the dictionary.

Step 5. When the symbol list for that user appears, choose **Edit > Paste Dict**.

Step 6. Remember to commit your transaction before logging out.

Figure 5.6 The Symbol List Browser



To Examine a User's Group Memberships

No privileges are required for this operation.

Step 1. In the **GemStone** menu, choose **Admin > Users**.

Step 2. When the GemStone User List (Figure 5.4 on page 5-16) appears, select the UserId, then click **Show User Info**.

Step 3. The selected user's group memberships are listed in the GemStone User dialog.

To Add a User to a Group

Privileges required: OtherPassword and write authorization to the segment of the users's UserProfile.

You can use the GemStone User dialog to add group membership for a particular user:

- If the group already exists in the repository, click **Add To Group** in the GemStone User dialog. Then choose the group name in the list that appears.
- If the group does not yet exist in the repository, click **Add To New Group** in the GemStone User dialog. Then enter the name in the dialog box that appears.

To Remove a User from a Group

Privileges required: OtherPassword and write authorization to the segment of the users's UserProfile.

You can use the GemStone User dialog to remove group membership for a particular user.

Select the group name in the list of group memberships, then click **Remove From Group**. When you're done, click either **OK** or **Apply**.

Administering Segment Authorizations

This section tells how to administer segments using GemBuilder's Segment Tool.

To Find Out Who Is Authorized to Read or Write in a Segment

Privileges required: read authorization for the segment with which this segment itself is associated, such as the DataCurator Segment.

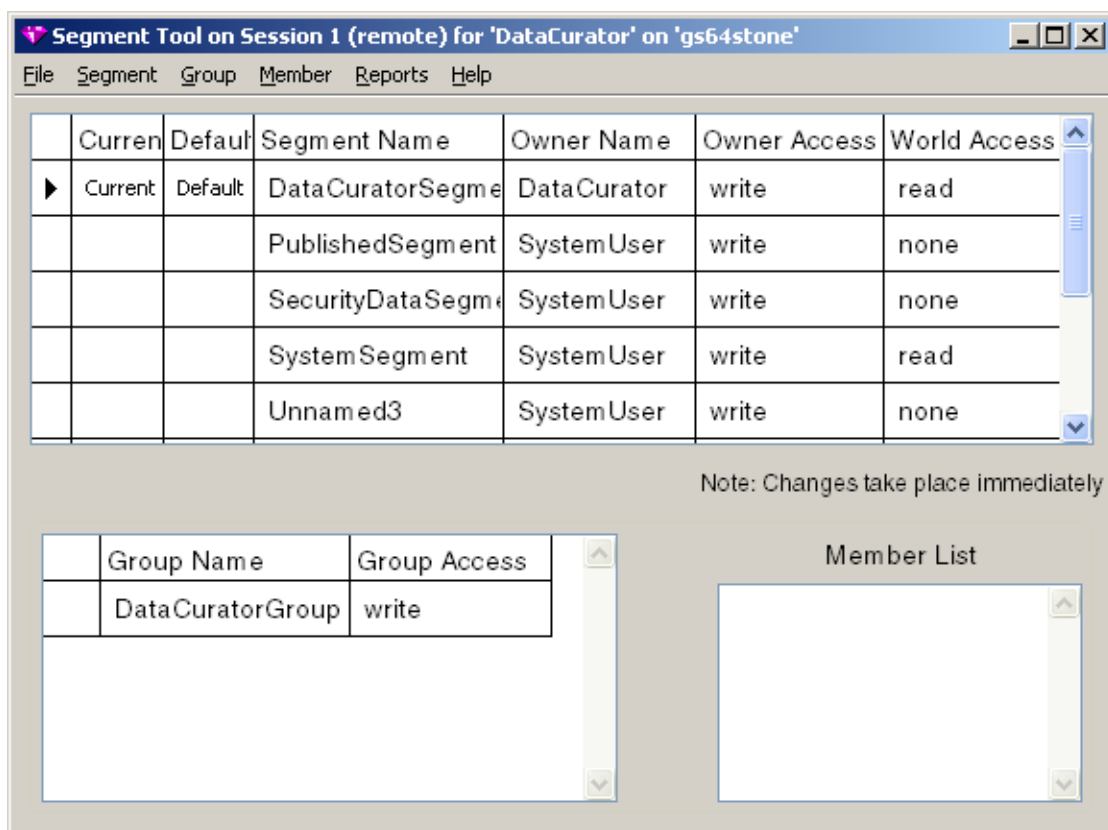
Each segment maintains access authorization for its owner, the world, and an unlimited number of groups. There are three levels of access: none, read-only, and write (which includes read access).

Step 1. In the **GemStone** menu, choose **Tools > Segment Tool** to bring up the Segment Tool, Figure 5.7. (You can also access this tool by choosing **Show Segments** in a GemStone User dialog.)

Step 2. In the Segment Tool, choose **Reports > Segment Report**. The report lists the owner, world, and group authorizations for each segment.

Step 3. To view the members of a particular group, choose **Reports > Group Report**. To view the groups to which each user belongs, choose **Reports > User Report**.

Figure 5.7 The Segment Tool



To Change the Authorization of a Segment

Privileges required: SegmentProtection or be the segment's owner, and write authorization to the DataCurator segment.

Each segment maintains access authorization for its owner, the world, and an unlimited number of groups. There are three levels of access: none, read-only, and write (which includes read access).

The new authorization will take effect for logins following the commit of the current transaction.

Step 1. In the GemStone menu, choose **Segments** to bring up the Segment Tool, Figure 5.7. (You can also access this tool by choosing **Show Segments** in a GemStone User dialog.)

The top half of the dialog shows the owner, the owner's access, and the world access for each segment in the repository.

Step 2. To change owner or world access, select the existing permission you want to change. Then enter a new permission ("read", "write", or "none").

Step 3. To set up or change group access to this segment, do the following:

- Make sure the segment is selected in the top half of the tool. If necessary, click in the first column of the segment's row.
- To add a group to the authorization list for this segment, choose **Add** from the **Group** menu. Enter the group name in the dialog box that appears. If the group does not exist in the repository, you will be asked to confirm its creation.
- To remove a group from the authorization list, first select the group by clicking in the first column of the **Group Name** list. Then choose **Remove** from the **Group** menu. You will be asked to confirm the action.
- To change the type of access for a particular group, first select that group in the **Group Name** list and select the existing permission. Then enter the new permission ("read" or "write").
- To add a member to a group that has access to this segment, first select that group in the groups list. Then choose **Add** from the **Member** menu. Enter the UserId and choose **OK**. (A UserProfile with that UserId must already exist in the repository.)
- To remove a member from a group that has access to this segment, select the UserId in the member list and choose **Remove** from the **Member** menu. You will be asked to confirm the action.

Step 4. Remember to commit your transaction before logging out. A convenient way to do that is by choosing **Commit** from this tool's **File** menu.

To Change a User's Default Segment

Privileges required: DefaultSegment to change your own, or write authorization in the DataCurator Segment to change another's; and write authorization to the DataCurator segment.

Changes to another user's default segment do not take effect until the next login.

To change your own default segment:

Step 1. Open the Segment Tool by choosing **Tools > Segment Tool** in the **GemStone** menu.

Step 2. Select the desired segment by clicking in its first column. Then choose **Segment > Make Default**. (Alternatively, you can type the @ symbol in front of the segment name.)

To change someone else's default segment:

Step 1. In the GemStone User dialog, choose **Show Segments**.

Step 2. When the Segment Tool appears, select the desired segment by clicking in the first column. Choose **Segment > Grab**.

Step 3. Return to the GemStone User dialog. Choose **Paste To Default Segment**.

Step 4. Choose **OK** or **Apply**.

NOTE

If you change any user's default segment (including your own) to a segment 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.

Step 5. Remember to commit your transaction before logging out.

5.5 Using Topaz for Administration

Performing system administration tasks using Topaz involves three steps:

1. Start Topaz and log in to GemStone.
2. Use the Topaz **printit** command to execute Smalltalk expressions.
3. Use the Topaz **commit** command to make your work a permanent part of the GemStone repository.

NOTE

You must commit your changes to the repository in order to make them permanent.

Logging in Through Topaz

To start Topaz, enter:

```
% topaz -l
```

Topaz announces itself with a banner:

```
-----
GemStone/S64 Object-Oriented Data Management System
Copyright (C) GemStone Systems, Inc. 1986-2007.
All rights reserved.
covered by Patent Number 6,567,905 Generational Garbage Collector.
-----
PROGRAM: topaz, Linear GemStone Interface (Remote Session)
VERSION: 2.2.0, Fri Mar 23 16:54:03 US/Pacific 2007
BUILD: gss64bit-15
BUILT FOR: SPARC (Solaris)
MODE: 64 bit
RUNNING ON: 2-CPU handel sun4u (Solaris 2.9 Generic_117171-08) 400MHz
sparcv9,2048MB
PROCESS ID: 3596      DATE: 03/26/07 15:30:39 PDT
USER IDS: REAL=gsuser (531) EFFECTIVE=gsuser (531)
-----
```

Next, set three parameters that allow you to log in to the repository: the name of your GemStone repository monitor, your GemStone user name, and your GemStone password. Use the Topaz **set** command to establish these parameters.

For example:

```
topaz> set gemstone gs64stone
topaz> set username DataCurator
topaz> set password
GemStone Password? (type your GemStone password)
topaz> login
[Info]: LNK client/gem GCI levels = 801/801
[Info]: User ID: SystemUser
[Info]: Repository: gs64stone
[Info]: Session ID: 5
[Info]: GCI Client Host: <Linked>
[Info]: Page server PID: -1
[Info]: Login Time: 03/10/07 15:04:10 PST
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1>
```

The session number in the topaz prompt (“topaz 1>”) is a reminder that you are logged in to a GemStone session. If you log in to additional sessions, the number in the prompt shows which session is active.

If you want to return to your host operating system from Topaz without terminating your GemStone session, use the Topaz **spawn** command. When the operating system prompt appears, you can execute system commands. When you’re finished, type **exit**. You can then resume your GemStone work from within Topaz.

The Printit Command

The **printit** command sends text following the command to GemStone for execution as Smalltalk code, and displays the results. (The **run** command does the same thing.) If there is an error in your code, Topaz displays an error message instead of a legitimate result.

Smalltalk text is terminated by the first line that contains a % symbol in column 1. For example:

```
topaz 1> printit
2 + 2
%
4
```

For complete information about Topaz commands, see the *GemStone Topaz Programming Environment* manual. You can also use the **help** command at the Topaz prompt.

The Commit Command

The **commit** command ends the current transaction and stores your changes in the repository.

NOTE

*Always remember to commit changes you want to become persistent. Your changes are not part of the repository (and therefore not visible to subsequent sessions) until you issue the Smalltalk message **System commitTransaction** or invoke the Topaz **commit** command.*

Administering User Accounts

This section explains how to create user accounts (UserProfiles) and how to examine, modify, and remove existing accounts using Topaz. Most of these tasks explicitly access UserProfiles in the global object AllUsers.

To 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> level 1
topaz 1> printit
AllUsers collect: [:each | each userId ].
%
an IdentityBag
  _varyingSize      5
  _numEntries       5
  _indexedPaths     nil
  _levels           0
  #1 DataCurator
  #2 SystemUser
  #3 SymbolUser
  #4 GcUser
  #5 Nameless
```

To Add a User

Privileges required: OtherPassword.

This section shows how to create a new GemStone UserProfile object. The new UserProfile is stored in the global object AllUsers (a UserProfileSet), along with all other UserProfiles.

NOTE

You must add each new user to AllUsers.

In addition to creating the new UserProfile, you should also see that each user's UNIX environment is set up to provide access to GemStone. That discussion is on page 5-17.

GemStone Smalltalk provides two methods for adding a user. The simplified form requires only a UserId and a password. The complete form also allows you to set the user's privileges and group memberships.

Without Privileges or Groups

You can use the simplified form to create a new user with no privileges or group memberships.

In the following example, you must supply the new user's userId and password (each as a String). The password must not be the same as the UserId and must not be longer than 1024 characters.

This method creates and commits a new segment and makes this the default segment for the new user.

```
topaz 1> printit
AllUsers addNewUserWithId: 'theUserId'
        password: 'thePassword' .
"commit the new UserProfile"
System commitTransaction
%
```

To Assign Privileges and Group Memberships

Using the complete form allows you to assign privileges to the new user and add the user to groups.

Execute the following expression:

```
AllUsers addNewUserWithId: 'theUserId'  
  password: 'thePassword'  
  defaultSegment: nil  
  privileges: anArrayOfPrivStrings  
  inGroups: aCollectionOfGroupStrings
```

You must supply the new user's UserId and Password, and specify any privileges or group memberships. The password must not be the same as the UserId and must not be longer than 1024 characters.

For example:

```
topaz 1> printit  
AllUsers addNewUserWithId: 'Mary'  
  password: 'herPasswd'  
  defaultSegment: nil  
  privileges: #( 'UserPassword' )  
  inGroups: #( 'MarathonRunners' ).  
"commit the new UserProfile"  
System commitTransaction.  
%
```

For more information about privileges, see the discussion entitled "UserProfiles" on page 5-3.

To Change Your Own Password

Privileges required: UserPassword.

Your choice of passwords for your own account may be subject to optional constraints as to pattern and the use of certain words. For information, ask your system administrator or see "How to Configure GemStone Login Security" on page 5-42.

To modify your own GemStone password, execute the following expression. The new password will take effect when you commit the current transaction. The

password must not be the same as the UserId and must not be longer than 1024 characters.

```
topaz 1> printit
System myUserProfile
      oldPassword: 'oldPasswordString'
      newPassword: 'newPasswordString' .
System commitTransaction
%
```

To Change Another User's Password

Privileges required: OtherPassword.

To modify the password of a GemStone user (other than your own), execute the following expression.

```
topaz 1> printit
(AllUsers userWithId: 'theUserId')
      password: 'newPasswordString' .
System commitTransaction
%
```

The new password takes effect when you commit the current transaction.

The password set by this method is not subject to the constraints described on page 5-42 because it can only be set 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.

To Examine 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, first make sure that the Topaz display level is sufficient to display that information. The Topaz display level determines the amount of detail that appears in the results of

Smalltalk execution. Use the Topaz **level** command to raise the level to at least 1, so that privileges information will be displayed:

```
topaz 1> level 1
```

Now send the following message to the desired user's UserProfile:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId') privileges
%
```

This message returns an Array of Strings. Each String in the Array corresponds to one of the user's privileges. Table 5.1 (on page 5-5) lists the Smalltalk methods that correspond to each privilege.

To Assign a Privilege to a User

Privileges required: OtherPassword and write authorization to the segment 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:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId')
    addPrivilege: aPrivilegeString .
System commitTransaction
%
```

Here's an example that assigns three new privileges to user Bob:

```
(AllUsers userWithId: 'Bob')
    addPrivilege: 'SystemControl';
    addPrivilege: 'SessionAccess';
    addPrivilege: 'UserPassword' .
System commitTransaction
%
```

To Revoke a User's Privilege

Privileges required: OtherPassword and write authorization to the segment 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:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId')
    deletePrivilege: aPrivilegeString .
System commitTransaction
%
```

The following example revokes three of user Jane's privileges:

```
(AllUsers userWithId: 'Jane')
    deletePrivilege: 'SystemControl';
    deletePrivilege: 'SessionAccess';
    deletePrivilege: 'UserPassword' .
System commitTransaction
%
```

To Redefine a User's Privileges

Privileges required: OtherPassword and write authorization to the segment of the user's UserProfile.

The new privileges will take effect when you commit the current transaction.

To redefine a user's privileges, perhaps adding some and revoking others, execute the following expression:

```
(AllUsers userWithId: theUserId) privileges: anArrayOfStrings
```

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
%
```

To Add 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 .
System commitTransaction
%
```

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.

To Add a SymbolDictionary to Someone Else's Symbol List

Privileges required: OtherPassword and write permission to the segment 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
%
```

To Remove 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
%
```

To Remove a SymbolDictionary from Someone Else's Symbol List

Privileges required: OtherPassword and write permission to the segment 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
%
```

To Examine a User's Group Memberships

No privileges are required for this operation.

To find out which groups a user belongs to, first make sure that the Topaz display level is sufficient to display that information. Use the Topaz **level** command to raise the level to at least 1, so that group membership information will be displayed:

```
topaz 1> level 1
```

Now execute the following expression:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId') groups
%
```

This expression returns a Set of Strings indicating the groups to which the user belongs.

To Add a User to a Group

Privileges required: OtherPassword and write authorization to the segment of AllUsers.

To add a user to a group, do the following:

```
topaz 1> printit
" If this is a new group, add it to the
  'master list' AllGroups "
('MarathonRunners' in: AllGroups)
    ifFalse: [AllGroups add: 'MarathonRunners' ].
(AllUsers userWithId: 'theUserId') addGroup: 'MarathonRunners'.
System commitTransaction
%
```

This expression adds the user to the group MarathonRunners by adding the group name to the list of groups maintained in the UserProfile. (This action takes effect when you commit the current transaction.)

If the group MarathonRunners did not previously exist, this expression creates it in AllGroups (the “master list” of all group names). See Appendix D, “GemStone Kernel Objects,” for more information about AllGroups and other predefined system objects.

To Remove a User from a Group

Privileges required: OtherPassword and write authorization to the segment of AllUsers.

To remove a user from a group, execute an expression of the following form:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId') removeGroup: 'Sprinters' .
System commitTransaction
%
```

This expression removes the designated group from the list of groups to which the user belongs. This action will take effect when you commit the current transaction. For more information about groups, see the Security chapter of the *Programming Guide for GemStone/S 64 Bit*.

To List All Members of a Group

No privileges are required for this operation.

To list all members of a user group, first make sure that the Topaz display level is sufficient to display that information. Use the Topaz **level** command to raise the level to at least 1, so that group membership information will be displayed:

```
topaz 1> level 1
```

Now execute the following expression:

```
topaz 1> printit
AllUsers membersOfGroup: aString
%
```

This expression returns an IdentitySet containing the userId for each member of the group.

To Remove a User Group

Privileges required: OtherPassword and write authorization to the segment of AllUsers.

To remove a user group from the global object AllGroups, execute the following expression. (You do not need to enter the comments, which are within double quotes.)

```
topaz 1> printit
| theGroup |
theGroup := aGroupString .
" Does the group still contain any members? If so, first
remove each member from the group "
(AllUsers usersInGroup: theGroup) do:
    [:aUserProfile| aUserProfile removeGroup: theGroup].
" It's OK to remove the group itself "
AllGroups remove: theGroup .
System commitTransaction
%
```

To Modify Someone's User ID

Privileges required: OtherPassword and write authorization to the segment of AllUsers.

The new user ID will take effect when you commit the current transaction.

To modify the user ID of a GemStone user (other than your own), execute the following expression:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId') userId: 'newId' .
%
```

An error is raised if *newId* is the *userId* of an existing UserProfile.

To Remove an Account

Privileges required: OtherPassword.

The global object AllUsers (a UserProfileSet) serves as the master list of all authorized GemStone users. When you need to cancel a user's access to GemStone, you can simply move that user's UserProfile from AllUsers to a UserProfileSet called OldUsers, which contains all obsolete UserProfiles. Any objects owned by members of OldUsers remain intact, but their owners can no longer access the repository.

First, verify that OldUsers already exists:

```
topaz 1> object OldUsers
```

If OldUsers already exists, Topaz will print some information about it (depending upon the current display level). If OldUsers does **not** already exist, Topaz will issue a message of the form `could not find an object named OldUsers`. To create OldUsers, execute the following expression:

```
topaz 1> printit
UserGlobals at: #OldUsers put: UserProfileSet new
%
```

Now add the obsolete UserProfile to OldUsers, then delete it from AllUsers:

```
topaz 1> printit
OldUsers add: (AllUsers userWithId: 'theUserId').
AllUsers remove: (AllUsers userWithId: 'theUserId')
      ifAbsent: [] .
System commitTransaction
%
```

To subsequently access any objects owned by the former user, you can refer to `(OldUsers userWithId: 'theUserId')` wherever you would refer to an active UserProfile.

Administering Segment Authorizations

This section tells how to administer segments using Topaz.

To Find Out Who Is Authorized to Read or Write in a Segment

Privileges required: read authorization for the segment with which this segment is associated, such as the DataCurator Segment.

Each segment 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).

You can find out who is authorized to read or write objects in a segment by sending it the message `asString`. For instance:

```
topaz 1> printit
PublishedSegment asString
%
aSegment, Number 5 in Repository SystemRepository
Owner SystemUser write
Group Subscribers read
Group Publishers write
World none
```

To Change the Authorization of a Segment

Privileges required: SegmentProtection or be the segment's owner, and write authorization to the DataCurator segment.

Each segment maintains access authorization for its owner, the world, and an unlimited number of groups. There are three authorization symbols: #none, #read (read-only), and #write (which includes read permission).

The new authorization will take effect for logins following the commit of the current transaction.

CAUTION

Do not, under any circumstances, attempt to change the authorization of the SystemSegment.

To change the authorization for a segment, execute any (or all) of the following expressions.

```
topaz 1> printit
theSegment ownerAuthorization: #anAuthorizationSymbol .
%
topaz 1> printit
theSegment worldAuthorization: #anAuthorizationSymbol .
%
topaz 1> printit
theSegment group: #aGroupString
authorization: #anAuthorizationSymbol .
%
```

NOTE

Exercise caution when changing the authorization for any segment that a user may be using as his or her default segment or current segment — whether or not the user owns the affected segment. If a user attempts to

commit a transaction, but has created objects in a segment 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 segment, you could execute the following expression:

```
topaz 1> printit
(AllUsers userWithId: 'Eli') defaultSegment
  group: #Accounting authorization: #read .
%
```

If the group #Accounting does not exist, GemStone will return an error. The discussion "Add a User to a Group" earlier in this chapter tells how to create a new GemStone group.

To Remove a Group from a Segment's Authorization List

Privileges required: SegmentProtection, and write authorization for the segment.

To remove a group from a segment's list of authorized groups, execute the following expression:

```
topaz 1> printit
theSegment group: #aGroupString authorization: #none
%
```

To Change a User's Default Segment

Privileges required: DefaultSegment to change your own, or write authorization in the DataCurator Segment to change another's; and write authorization to the DataCurator segment.

Changes to another user's default segment do not take effect until the next login.

To change a user's default segment, execute the following expression:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId') defaultSegment: aNewSegment
%
```

NOTE

If you change any user's default segment (including your own) to a segment 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.

5.6 How to Configure GemStone Login Security

GemStone provides several login security features. 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 SystemUser, DataCurator, SymbolUser, and GcUser accounts are never disabled by the security features.

To Constrain the Choice of Passwords

Privileges required: OtherPassword and write authorization to the segment of AllUsers.

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 administrator

actions changing any other account's password using the OtherPassword privilege, and they do not invalidate existing passwords.

Table 5.2 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 5.2 Ways to Constrain the Password Pattern

Message to AllUsers	Comments
minPasswordSize: <i>aPositiveInteger</i>	Sets the minimum number of characters in a new password; 0 means no constraint.
maxPasswordSize: <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.)
maxRepeatingChars: <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.
maxConsecutiveChars: <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.
maxCharsOfSameType: <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.

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 5.2).

For example, to determine the current minimum size for a password:

```
topaz 1> printit
AllUsers minPasswordSize
%
6
```

Disallowing Particular Passwords

Privileges required: OtherPassword and write authorization to the segment of AllUsers.

You can create a list of disallowed passwords by adding Strings to the AllUsers instance variable `disallowedPasswords`. Any messages understood by class Set can be used. For instance:

```
topaz 1> printit
(AllUsers disallowedPasswords)
  addAll: #( 'Mother' 'apple pie' ) .
System commitTransaction
%
```

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`, the error `#rtRestrictedPassword` is returned.

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 the segment of AllUsers.

You can prevent each user from choosing the same password more than once by setting the AllUsers instance variable `disallowUsedPasswords` to true. For example:

```
topaz 1> printit
AllUsers disallowUsedPasswords: true .
System commitTransaction
%
```

The default setting 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, the error `#rtUsedPassword` is returned.

Clearing a User's Disallowed Old Passwords

Privileges required: OtherPassword.

You can clear the set of old passwords so that they can be reused by sending the message `clearOldPasswords` to that user's `UserProfile`. 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 account `qa2`:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') clearOldPasswords .
System commitTransaction
%
```

To Require Periodic Password Changes

Privileges required: OtherPassword and write authorization to the segment of `AllUsers`.

You can require users to change their password periodically by sending the message `UserProfileSet>>passwordAgeLimit: numberOfHours`. For example, to set the limit to 120 days:

```
topaz 1> printit
AllUsers passwordAgeLimit: 120 * 24 .
System commitTransaction
%
```

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 GemStone security log (currently, the Stone's log).

If a user does not change the account's password within the specified interval, the account is disabled. Attempts to log in return error `#gsErrLoginFailure`. However, the `SystemUser`, `DataCurator`, `SymbolUser`, and `GcUser` accounts are never disabled by password aging.

DataCurator or another user with the OtherPassword privilege can reactivate the disabled account by giving it a new password. For details about how to do this, see page 5-20 (with GemBuilder) or page 5-31 (with Topaz).

Providing Warning of Password Expiration

Privileges required: OtherPassword and write authorization to the segment of AllUsers.

You can provide an automatic warning to users whose password is about to expire by sending the message

UserProfileSet>>passwordAgeWarning: *numberOfHours*. For example, to warn users who log in within five days of the time their password will expire, do this:

```
topaz 1> printit
AllUsers passwordAgeWarning: 5 * 24 .
System commitTransaction
%
```

Logins within *numberOfHours* prior to expiration receive the error #rtErrPasswordExpirationWarning.

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
%
03/03/07 11:28 AM
```

To Disable Inactive Accounts

Privileges required: OtherPassword and write authorization to the segment of AllUsers.

You can have the system disable accounts for which there has been no login for a specified length of time. To do this, send the message `staleAccountAgeLimit: numberOfHours` to AllUsers. This example disables accounts when they have not logged in for 30 days:

```
topaz 1> printit
AllUsers staleAccountAgeLimit: 30 * 24 .
System commitTransaction
%
```

Each time this method is invoked, the action is recorded in the GemStone security log (currently, the Stone's log).

A setting of 0 (the default) disables account aging.

The SystemUser, DataCurator, SymbolUser, and GcUser accounts are not disabled by this mechanism.

DataCurator or another user with the OtherPassword privilege can reactive the account by giving it a new password. For details about how to do this, see page 5-20 (with GemBuilder) or page 5-31 (with Topaz).

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: 'qa2') lastLoginTime US12HrFormat
%
03/03/07 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`: `passwordAgeLimit`, `passwordAgeWarning`, or `staleAccountAgeLimit`.

To 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 "qa2" to two more logins under the current password:

```
topaz 1> printit
(AllUsers userWithId: 'qa2')
  loginsAllowedBeforeExpiration: 2 .
System commitTransaction
%
```

A setting of 0 (the default) disables this feature.

The limit remains in effect until the user changes the password (see pages 5-20 and 5-31). 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 account by giving it a new password. For details, see page 5-20 (with `GemBuilder`) or page 5-31 (with `Topaz`).

The `SystemUser`, `DataCurator`, `SymbolUser`, and `GcUser` accounts are not disabled by this mechanism.

To Limit Concurrent Sessions by a Particular UserId

Privileges required: OtherPassword.

You can limit the number of concurrent sessions logged in under a particular `userId` by sending the message `activeUserIdLimit: aPositiveInteger` to the `UserProfile` for that account. For example, the following limits the `userId` “qa2” to four concurrent sessions:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') activeUserIdLimit: 4 .
System commitTransaction
%
```

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 the fatal error `#gsActiveUserLimitExceeded` is returned.

To Record Login Failures

The Stone repository monitor keeps track of login failures (incorrect passwords) for each account and can write that information to the GemStone security log (currently, 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` and `STN_LOG_LOGIN_FAILURE_TIME_LIMIT` configuration options (see page A-30).

The log message gives the following information:

```
---Fri 14 Jan 2007 09:39:40 PST ---
GemStone user DataCurator has failed on 10 attempt(s)
to log in within 1 minute(s).
The last attempt was from user account writer1 on host
name docs.
```

Disabling Further Login Attempts

If login failures continue, the Stone repository monitor can disable the account by changing the GemStone password to an invalid one (that is, to a password that cannot be entered). 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` and `STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT` configuration options (page A-26).

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 SystemUser, DataCurator, SymbolUser, and GcUser accounts are not disabled by this mechanism.

To reactivate an account that has been disabled by this mechanism, the DataCurator (or another account with explicit OtherPassword privilege) must change the account's password to a valid one. See the instructions under "To Change Another User's Password" on pages 5-20 and 5-32.

To Find Out Which Accounts Have Been Disabled

Privileges required: OtherPassword.

The message `AllUsers findDisabledUsers` returns a SortedCollection of UserProfiles that are disabled by one of the security precautions discussed above:

- The password expired (through aging or a login limit).
- The account remained inactive.
- There were repeated password failures.

For example:

```
topaz 1> level 1
topaz 1> printit
AllUsers findDisabledUsers
  collect: [:aUser | aUser userId ] .
%
an Array
  #1 qa2
  #2 qa3
```

In each case, the account has been disabled by setting its password to one that is invalid. DataCurator or another user with the OtherPassword privilege can reactivate an account by giving it a new password. For information about how to do that, see pages 5-20 and 5-31.

To Verify That an Account Is Disabled

Privileges required: OtherPassword.

You can verify that a particular account is disabled by sending the message `isDisabled` to the account's `UserProfile`. The method returns either `True` or `False`. This example inquires about account `qa2`:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') isDisabled
%
true
```

To Find Out Why an Account Was Disabled

Privileges required: OtherPassword.

You can find out why a particular account was disabled by sending the message `reasonForDisabledAccount` to the account's `UserProfile`. This example inquires about account `qa2`:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') reasonForDisabledAccount
%
LoginsWithSamePassword
```

The value returned is one of these Strings: 'PasswordAgeLimit', 'StaleAccount', 'LoginsWithSamePassword', or 'LoginsWithInvalidPassword'.

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. Within GemStone Smalltalk, the repository is the single instance of Class Repository. Initially, it has the name `SystemRepository`.

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. Chapter 1 explains how this mapping is done through GemStone configuration options. Initially, the repository is contained in a single file, `$GEMSTONE/data/extent0.dbf`.

Whenever GemStone performs a *checkpoint*, it makes sure that transactions committed before the checkpoint have been written to the repository extents. The `STN_CHECKPOINT_INTERVAL` configuration option (page A-25) sets the maximum time between checkpoints. (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 “To Control Checkpoint Frequency” on page 1-42.

This chapter explains how the repository grows, and tells you how to perform a number of administrative tasks related to the repository:

- How to determine the amount of free space in the repository (page 6-3)
- How to create more space by adding an extent while the repository is in use (page 6-5)
- How to remove an extent (page 6-7)
- How to reallocate objects among extents (page 6-8)
- How to recover from an error caused by a full disk (page 6-14)

6.1 Repository Growth

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 1 MB or more.

- Committed objects are flushed to the disk at certain times by writing the new page during a *checkpoint* of the repository. Private (invisible) objects, such as the structure that supports large collections, also are part of the repository. (Committed changes are written immediately to a transaction log to preserve the information in case of a system failure.)
- Allow at least 10 MB per session for repository space. If that space isn't available, the repository monitor will expand the extent to provide the necessary free space.
- The `STN_FREE_SPACE_THRESHOLD` configuration option (page A-27) sets the minimum amount of free space to be available in the repository (the default is 1 MB). If free space falls to that threshold, the Stone repository monitor enlarges the repository.

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 10, "Managing Growth," for a discussion of garbage collection in GemStone.

6.2 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 command `ls -l extentName`. For example:

```
topaz 1> printit
SystemRepository fileSize
%
23068672
```

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
%
11272192
```

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`.)

For example:

```
topaz 1> printit
SystemRepository fileSizeReport
%
Extent #1
-----
Filename = !TCP@nodename#dbf!/users/extents/extent0.dbf

File size =      22.00 Megabytes
Space available =  5.56 Megabytes

Extent #2
-----
Filename = !TCP@nodename#dbf!/users/extents/extent1.dbf

File size =      6.00 Megabytes
Space available =  0.12 Megabytes

Totals
-----
Repository size = 28.00 Megabytes
Free Space =     5.69 Megabytes
```

The amount of free space in the repository can also be determined from the cache statistic FreePages (see page 8-33). To obtain the free space, multiply FreePages by the page size, 16384.

6.3 How to Add 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` and `DBF_EXTENT_SIZES` configuration options (page A-12). 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 specification (a file or a raw partition) to the configuration file for the Stone. This section describes the Smalltalk methods that allow you to do this.

For general information about multiple extents, see “To Configure the Repository Extents” on page 1-19.

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. 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 3/18/07
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` option at system startup (page A-13). Because extents created with this method have no maximum size, 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. (For a discussion of weighted allocation, see page 1-19.)

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 the configuration file option `DBF_PRE_GROW` is set to True (page A-13), 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.

6.4 How 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 on page 9-8.

You cannot use an online or offline extent backup to shrink extents.

Step 2. Shut down the Stone repository monitor.

Step 3. Modify the `DBF_EXTENT_NAMES` configuration parameter (page A-12) to show the new extent structure.

Step 4. Restore the repository from your Smalltalk full backup. Follow the GemStone restore procedure described on page 9-14.

6.5 How To Reallocate 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 `DBF_ALLOCATION_MODE` 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 in the `DBF_ALLOCATION_MODE` configuration option (page A-12).

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 on page 9-8.

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 the `DBF_ALLOCATION_MODE` configuration option to reflect the intended distribution of pages (see “Allocating Data to Multiple Extents” on page 1-23). For example:

```
DBF_ALLOCATION_MODE = 10, 10 ;
```

Step 5. Restore the repository from your Smalltalk full backup, using the procedure beginning on page 9-14.

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 *GemStone/S 64 Bit Programming Guide*.

To Reallocate Objects Among the Same Number of Extents

Changes to `DBF_ALLOCATION_MODE` (page A-12) 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 on page 6-8) or you can specify a maximum size on the extent files, 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 1 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 on page 9-8.

Step 2. Shut down the Stone repository monitor.

Step 3. Edit the `DBF_EXTENT_SIZES` configuration option (page A-13) to limit the size of the first extent temporarily to 200 MB. For example:

```
DBF_EXTENT_SIZES = 200, , ;
```

Step 4. Edit the `DBF_ALLOCATION_MODE` configuration option (page A-12) to reflect the intended distribution of pages (see “Allocating Data to Multiple Extents” on page 1-23). 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 beginning on page 9-14.

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 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, refer to the *GemStone/S 64 Bit Programming Guide*.

6.6 How to Shrink 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. For example:

```
topaz 1> printit
SystemRepository markForCollection
%
```

For further information about this method, see “`markForCollection`” on page 10-26.

Alternatively, you can run the FDC/MGC two-step process, which performs the same function as `markForCollection` but may be more suitable for large production systems. For details, see page 10-31.

Step 2. Wait for GemStone to complete the garbage collection and reclaim the space. The time required depends on several factors: the size of the repository, the number of Reclaim GcGems currently running, and (in multi-user mode) the status of other sessions. For details, see “Reclaiming Pages” on page 10-35.

If other users are logged in, space will not be reclaimed until all sessions have committed or aborted any transactions concurrent with the

markForCollection or FDC/MGC process. For further information, see “To Identify Sessions Holding Up Page Reclamation” on page 10-41.

Step 3. Make a Smalltalk full backup of your repository by sending it the message `fullBackupTo:fileOrDevice MBytes:byteLimit`.

For example:

```
topaz 1> printit
SystemRepository fullBackupTo: '/users/bk/oct31'
  MBytes: 0
%
```

This example writes the backup to a single disk file. If you need to write multiple files or multiple tapes, see “To Create a Backup in Multiple Files” on page 9-12.

Step 4. Take the repository offline:

```
topaz 1> printit
System shutDown
%
```

Step 5. 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 6. To put the repository back online, issue the `startstone` command:

```
% startstone gemStoneName
```

If you do not specify `gemStoneName`, `startstone` defaults to `gs64stone`.

Step 7. 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 8. Restore the repository by using the method

Repository>>restoreFromBackup : *fileOrDevice*, using the same file or device as in Step 3. 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/bk/oct31'
%
```

If you need to restore multiple files or multiple tapes, use the method Repository>>restoreFromBackups : *fileOrDeviceArray* instead:

```
topaz 1> printit
SystemRepository restoreFromBackups:
#( '%/backups/July_1.1'
    '%/backups/July_1.2' )
%
```

(For more information, see “To Restore Multiple-File Backups” on page 9-23.)

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 30569 objects
restored and 0 corrupt objects not restored.
```

Each restore operation, on completion, terminates its GemStone session. You will need to log in again before performing the next restore operation.

Step 9. Between the time the full backup was started (Step 3) and the time the repository was shut down (Step 4), 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. [null]

topaz> login
gci login: currSession 1 rpc gem processId -1
successful login

topaz 1> printit
SystemRepository commitRestore
%
Restore from transaction log(s) succeeded. commitRestore
succeeded
```

6.7 How to Check Page Fragmentation

Space within the repository is managed in pages having a fixed size of 8 KB. It is possible for these pages to become *fragmented*—that is, only partially filled with objects. GemStone automatically schedules reclamation 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)
- 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

6.8 How to Recover 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 "How to Recover from Tranlog-Full Conditions" on page 7-13

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 (page A-27) 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 sends error `#rtErrFreeSpaceThreshold` 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 (page A-27), 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:*extentFileName* or
createExtent:*extentFileName* withMaxSize: *aSmallInteger*. If the Stone has stopped, you can edit the parameters in the configuration file before restarting it. See “How to Add Extents” on page 6-5.

Managing Transaction Logs

7.1 Overview

A transaction log contains the information necessary to redo transactions to the repository that have been committed by GemStone/S 64 Bit 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 the optional full-logging mode can be used to recreate all transactions committed since the most recent backup was written.

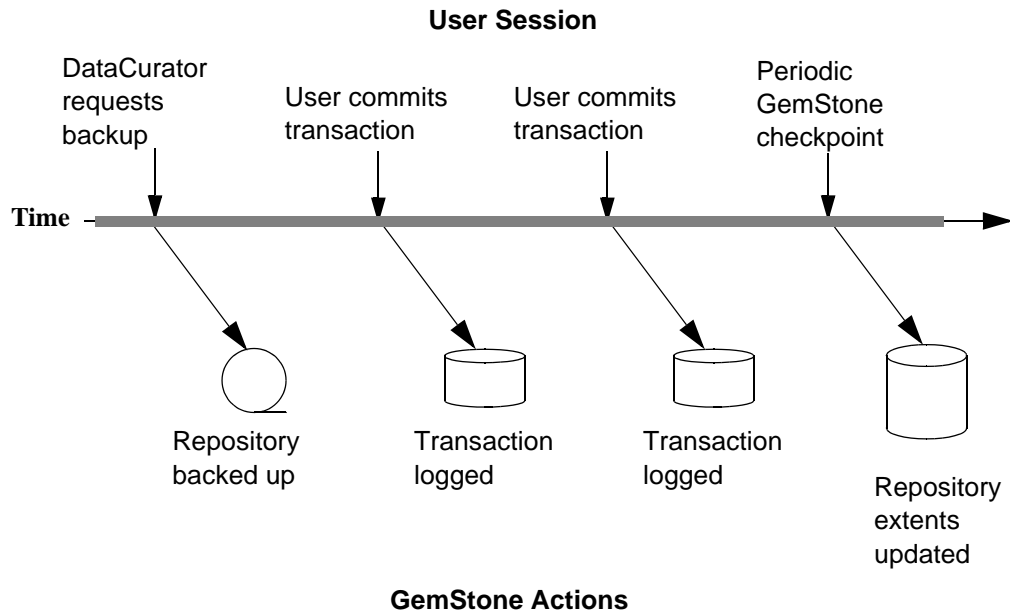
The transaction log is implemented as a sequence of files having names of the form `tranlog0.dbf ... tranlogNNN.dbf`. The numeric `fileId` starts at 0 when the Stone starts with a copy of the initial repository extent (`$GEMSTONE/bin/extent0.dbf`), and there are no existing transaction logs with that `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. You can control the filename prefix by setting the `STN_TRAN_LOG_PREFIX` configuration option (page A-36).

These logs are written to a list of directories (or raw partitions) specified by the `STN_TRAN_LOG_DIRECTORIES` configuration option (page A-36), which is treated

as a circular list. Each log is limited to the size set for that directory or raw partition by `STN_TRAN_LOG_SIZES` (page A-37). 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 7.1 on page 7-4.) Collectively, the transaction log files logically form an extremely large sequential file with a maximum size of 4×10^6 GB.

Between checkpoints, GemStone writes each committed transaction to a transaction log (Figure 7.1).

Figure 7.1 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. 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 recommended for deployed GemStone systems.
- To allow a simple operation to run unattended for an extended period, set `STN_TRAN_FULL_LOGGING` to `False` (the initial setting). This mode, known as *partial logging*, provides limited backup that ordinarily permits automatic recovery from system crashes that do not corrupt the repository.

Table 7.1 compares full and partial transaction logging.

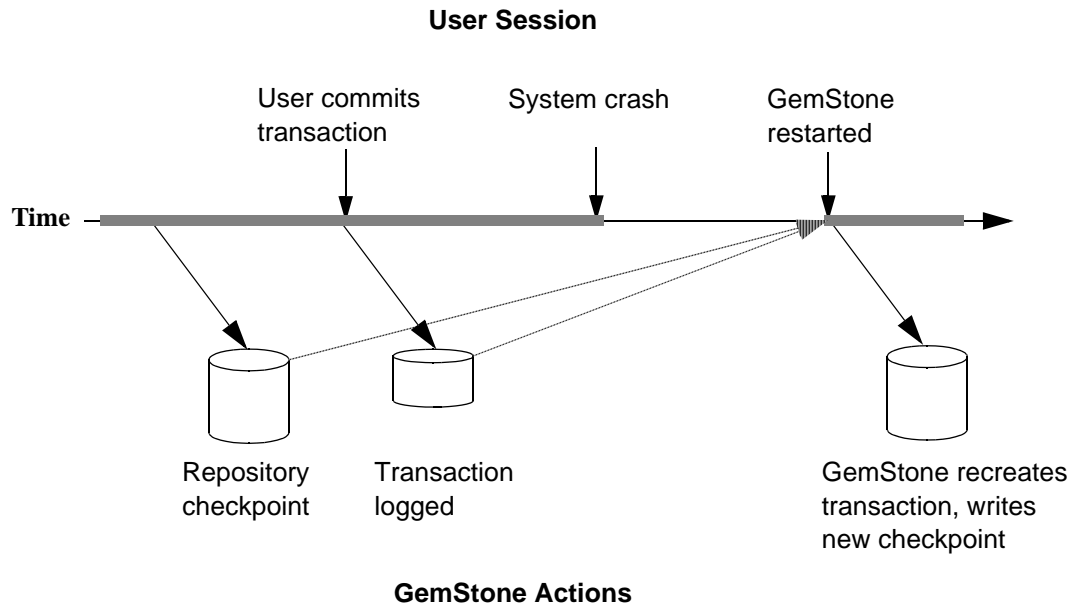
Table 7.1 Comparison of Full and Partial Transaction Logging

Characteristic	STN_TRAN_FULL_LOGGING=TRUE (Full Logging)	STN_TRAN_FULL_LOGGING=FALSE (Partial Logging)
Type of transaction logged	All transactions	Only those transactions smaller than STN_TRAN_LOG_LIMIT; successful commits of larger transactions cause an immediate checkpoint
Recovery from system crash (extents are OK)	Yes, automatic during restart using checkpoint and log	Yes, automatic during restart using checkpoint and log
Recovery of transactions since last backup (as after media failure)	Yes—can carry forward GemStone backup by recreating subsequently committed transactions	No—cannot recover transactions since the backup
Action when current log is full	Logging moves to the next directory or to the head of the list. If it is a file system directory, the Stone opens a new log file there; existing transaction logs are retained. If it is a raw partition, a new log can be opened only if the previous one has been archived and removed. The maximum number of file system logs online at one time depends on disk space. The maximum number of raw partition logs depends on the number of partitions listed in STN_TRAN_LOG_DIRECTORIES.	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

Recovering from an Unexpected Shutdown

In the event of a system crash, GemStone can recover by automatically reapplying transactions from the log to the latest checkpoint (Figure 7.2). This allows you to restart from where you were at the time of the crash.

Figure 7.2 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.

You can use the transaction logs to restore transactions committed since the last GemStone online extent backup or a Smalltalk full backup. The following steps show what you must do to prepare:

Step 1. Change the `STN_TRAN_FULL_LOGGING` configuration option to True.

Step 2. Restart GemStone.

Step 3. Make a GemStone online extent backup (page 9-3) or a Smalltalk full backup (page 9-8).

How the Logs Are Used

The GemStone restore procedure (Figure 7.3) starts by copying any good repository, preferably the initial repository extent that is distributed with GemStone. That repository contains the GemStone kernel classes and base system, which serve as a starting point for the restore.

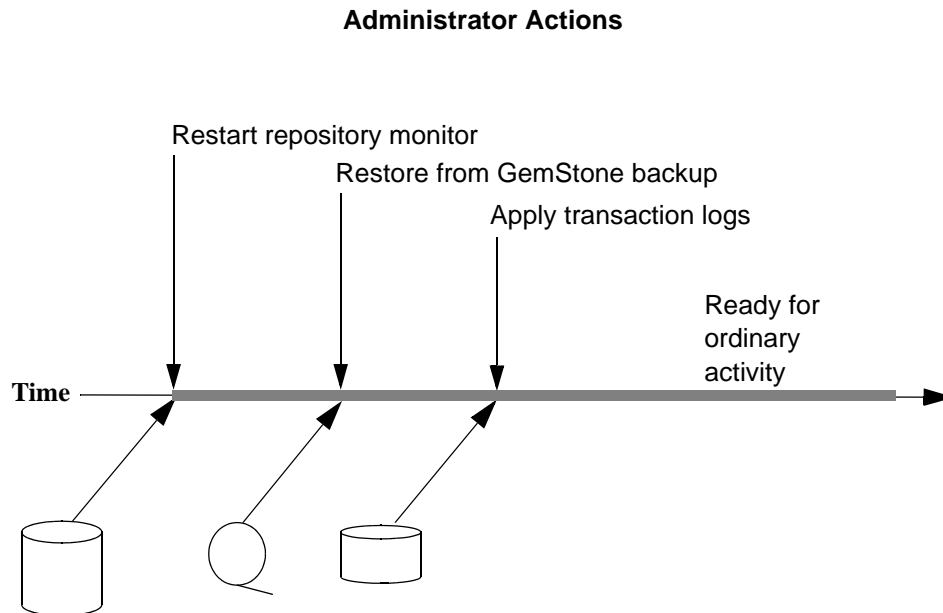
Next, you restore the full backup, using the appropriate procedure:

- To restore from an online extent backup, see page 9-7.
- To restore from a Smalltalk full backup, see page 9-14.

Finally, if the repository was in full transaction logging mode, you restore transactions committed since the backup by reading the transaction logs in the order in which they were generated. The restore procedures in Chapter 9 (listed above) tell how to restore transactions committed since the backup.

NOTE

Restoring a repository resets its origin to the time of the backup that was restored. Subsequent transactions can be restored only by starting with that backup or a more recent one.

Figure 7.3 Applying Transaction Logs to a GemStone Backup

7.2 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 B-10) 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 (page A-36).

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.

Both the **copydbf** and **removedbf** commands can be used with a remote node even if it is not running NFS (a NetLDI must be running on that node). For further information about **copydbf** and **removedbf**, see the command descriptions in Appendix B.

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)
  Last checkpoint written at: 03/31/07 15:30:30 PDT.
  Oldest tranlog needed for recovery is fileId 5 (
  tranlog5.dbf ).
```

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)
  The previous file last recordId is -1.
Destination file: /dev/null
  byteOrder: Sparc (MSB first)
  Full backup started from checkpoint at: 03/31/07 15:28:37
  PDT.
  Oldest tranlog needed for restore is fileId 5 (
  tranlog5.dbf ).
```

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: 8
%
```

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 A-37).

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 1-38. The Repository methods in Table 7.2 return other information that is helpful in managing transaction logs.

Table 7.2 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>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.
<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 till 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
%
9
```

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 Start Checkpoints

Privileges required: SystemControl.

Class System (category Transaction Control) 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, use the following procedure:

Step 1. Perform a Smalltalk full backup using `Repository>>fullBackupTo:.` See “How to Make a Smalltalk Full Backup” on page 9-8.

Step 2. Edit the configuration file to set the `STN_TRAN_FULL_LOGGING` option to False.

Step 3. Stop the Stone repository monitor.

Step 4. Replace the first (primary) extent file with a copy of `$GEMSTONE/bin/extent0.dbf`. Delete any other extent files.

Step 5. Restart GemStone.

Step 6. Restore the backup using `Repository>>restoreFromBackup: or restoreFromBackups:.` See “How to Restore from a Smalltalk Full Backup” on page 9-14.

7.3 How to Manage Partial Logging

Partial logging is GemStone's default mode because it provides ease of administration with 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* - `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`.

You should ensure that there always is sufficient disk space for at least two log files (their default size is 10 MB each), 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` setting to True (page A-35) and restart the Stone repository monitor.

CAUTION

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.

7.4 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 error `#rtErrTranlogDirFull` is sent to 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 the `STN_TRAN_LOG_DIRECTORIES` configuration option (page A-36). 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 7-10.

When transaction log space becomes available, waiting sessions can complete operations that were blocked.

Monitoring GemStone

This chapter tells you:

- Where to look for the log files that GemStone/S 64 Bit processes create
- How to audit the repository
- 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 impede garbage collection activity, resulting in undesirable repository growth, until you either commit or abort.

NOTE

For sessions that are not committing changes to the repository, we recommend that monitoring be done in manual transaction mode. For details on entering and using manual transaction mode, see page 4-24.

8.1 GemStone System Logs

In addition to transaction logs, GemStone creates three types of log files:

- Logs for GemStone object server processes (page 8-2)
- Logs for processes related to individual GemStone sessions (page 8-7)
- Logs for GemStone network server processes, NetLDIs (page 8-9)

If a GemStone server is running, you can use the **gsl** utility to locate its logs. Use **gsl -x** to display the location of the current log file for Stones, NetLDIs, and the shared page cache monitors.

The logs for the AIO page servers, Free Frame page servers, SymbolGem, Page Manager, and Admin and Reclaim GcGems are in the same location as the corresponding Stone's log.

WARNING

*The Stone writes several files to the /opt/gemstone/locks directory. To avoid system failure, do not remove these files manually. Use **gsl -c** to remove unnecessary files from the /opt/gemstone/locks directory.*

For more about the **gsl** command, see page B-7.

GemStone Server Logs

The Stone repository monitor and its child processes each create a log file in a single location. By default, the files are in `$GEMSTONE/data` and have a name beginning with the name of the repository monitor. Table 8.1 shows typical log names for a repository monitor with the default name of `gs64stone`. Log names for child processes also include the process id and a descriptive suffix.

Table 8.1 Representative Log Names for GemStone Server Processes

gs64stone.log	Stone repository monitor
gs64stone_14033admingcgem.log	Admin GcGem
gs64stone_2963pcmon.log	Shared page cache monitor
gs64stone_2967pgsvrff.log	Free Frame page server
gs64stone_2984pgsvraio.log	AIO page server
gs64stone_2987pagemanager.log	Page Manager
gs64stone_2992reclaimcgem0-6.log	Reclaim GcGem
gs64stone_2994symbolgem.log	Symbol Gem

Several factors can alter the name and location of these logs. The precedence is

1. A path supplied by **startstone -l logFile**. If *logFile* is relative (that is, not a complete path preceded by a /), *logFile* is created in a current directory. Logs for the child processes in Table 8.1 are placed in the same directory.
2. A path specified by the GEMSTONE_LOG environment variable. If *logFile* is relative (that is, not a complete path preceded by a /), *logFile* is created in a current directory. Logs for the child processes in Table 8.1 are placed in the same directory.
3. $\$GEMSTONE/data/gemStoneName.log$.

Stone Log

The log for the Stone repository monitor is cumulative across runs. 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 DUMP_OPTIONS is set to True (page A-14), 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 the GcGem(s), and the corresponding processId.
- Each #abortErrLostOtRoot sent to a Gem.
- Each suspension and resumption of logins.
- Certain changes to the login security system.

Admin GcGem Logs

Each time the Stone repository monitor starts a new administrative garbage collection (Admin GcGem) session process, a new log is created in the same location as the Stone's log. If the Admin GcGem exits normally, the current log is automatically removed. When GcUser logs in again, a new log is opened under a name that includes the process ID of the new Admin GcGem.

These logs show the startup value of the Admin GcGem parameters that are stored in GcUser's UserGlobals and any changes to them.

If you want to retain this log even when the Admin GcGem process exits normally, edit the shell script `$GEMSTONE/sys/rungcgem`. Make sure that the following line is not commented out:

```
unset GEMSTONE_CHILD_LOG
```

CAUTION

Any changes that you make to the `rungcgem` script will also apply to the Page Manager and to all other GcGems.

Shared Page Cache Monitor Log

The log for the shared page cache monitor is located in the same directory as the Stone's log and is for a particular process (in Table 8.1, it is for processId 2963). 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 that node. By default, this log is named `startshrpcmonPidNode.log`, where *Pid* is a process Id and *Node* is the name of the node. The default location is the home directory of the account that started the Stone.

Among the items included in the log for the shared page cache monitor are:

- 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).

Free Frame Page Server Log

The log for the repository monitor's private free frame page servers are located in the same directory as the Stone's log. Each log is for a specific free frame page server process and is automatically removed if the page server exits normally.

These logs ordinarily are not of interest unless they contain an error message.

If you want to retain this log even when a free frame page server process exits normally, edit the shell script `$GEMSTONE/sys/runpgsvrmain`. Make sure that the following line is *not* commented out:

```
unset GEMSTONE_CHILD_LOG
```

CAUTION

Any changes that you make to the `runpgsvrmain` script will also apply to all other free frame and AIO page server processes.

AIO Page Server Log

The logs for the repository monitor's private AIO page servers are located in the same directory as the Stone's log. Each log is for a specific page server process and is automatically removed if the page server exits normally.

These logs ordinarily are not of interest unless they contain an error message.

If you want to retain this log even when an AIO page server process exits normally, edit the shell script `$GEMSTONE/sys/runpgsvrmain`. Make sure that the following line is *not* commented out:

```
unset GEMSTONE_CHILD_LOG
```

CAUTION

Any changes that you make to the `runpgsvrmain` script will also apply to all other AIO and free frame page server processes.

Page Manager Log

The Page Manager log is located in the same directory as the Stone's log. This log is for a specific page manager process, and is automatically removed if the page manager exits normally.

These logs ordinarily are not of interest unless they contain an error message.

If you want to retain this log even when the Page Manager process exits normally, edit the shell script `$GEMSTONE/sys/rungcgem`. Make sure that the following line is not commented out:

```
unset GEMSTONE_CHILD_LOG
```

CAUTION

Any changes that you make to the `rungcgem` script will also apply to all GcGem processes.

Reclaim GcGem Logs

Each time a new Reclaim GcGem session process is started, a new log is created in the same location as the Stone's log. (For details about Reclaim GcGems, see Chapter 10, "Managing Growth.") If the Reclaim GcGem exits normally, the current log is automatically removed. When GcUser logs in again, a new log is opened under a name that includes the process ID of the new Reclaim GcGem.

These logs show the startup value of the Reclaim GcGem parameters that are stored in GcUser's UserGlobals and any changes to them.

A Reclaim GcGem acts on a range of extent files. The digits in the log file name represent the start and stop indices (starting with 0) of the range for this Reclaim GcGem. In Table 8.1, the log name (`gs64stone_2992reclaimgcgem0-6.log`) indicates that the Reclaim GcGem is running on the first seven extents of the repository (0-6).

If you want to retain this log even when a Reclaim GcGem process exits normally, edit the shell script `$GEMSTONE/sys/rungcgem`. Make sure that the following line is not commented out:

```
unset GEMSTONE_CHILD_LOG
```

CAUTION

Any changes that you make to the `rungcgem` script will also apply to the Page Manager and to all other GcGems.

Symbol Gem Log

The Symbol Gem log is located in the same directory as the Stone's log. This log is for a specific Symbol Gem process, but is not removed when the symbol Gem exits.

These logs are not ordinarily of interest unless they contain an error message.

Logs Related to Gem Sessions

Sessions frequently depend on NetLDI services to spawn one or more supporting processes. In each case, the NetLDI creates a log file that includes in its name the identity of the node on which the process is running. Typical processes are:

- A Gem session process to serve an RPC application (linked Gem session processes do not produce logs).
- A page server (for the session) to access a repository extent on the server node.
- A page server (for the Stone) to start or access a shared page cache on the client's node.
- A shared page cache monitor (for the Stone) to manage the cache on the client's node.

When the application is running on the same node as the Stone repository monitor, only the Gem session process is needed, and only then to serve an RPC application.

These log files ordinarily are located in the home directory of the account that owns the corresponding process. For the Gem session process and 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**.

Table 8.2 shows typical log names for session-related processes, given a Stone and repository on *node1* with a login from a Gem session process on *node2*.

Table 8.2 Typical Logs Supporting Gem Sessions

Typical Name	GemStone Process
gemnetobject27853 <i>node2</i> .log	Gem session process on <i>node2</i> (serves an RPC application)
pgsvrmain27819 <i>node2</i> .log	Page server on <i>node2</i> that the repository monitor uses to create and access its shared page cache on <i>node2</i>
startshrpcmon27820 <i>node2</i> .log	Shared page cache monitor on <i>node2</i>
pgsvrmain12397 <i>node1</i> .log	Page server on <i>node1</i> that the Gem session process uses to access the repository extents on <i>node1</i>

If a process shuts down normally, the log file is automatically removed. After an abnormal shutdown, any log files that remain can provide helpful information.

You can change the default location by setting `#dir` or `#log` in the `GEMSTONE_NRS_ALL` environment variable for the NetLDI itself or for individual clients (see “To Set a Default NRS” on page 3-15). Alternatively, when you log in to GemStone, you can specify a different network resource string (NRS) in your login parameters.

The log for a Gem session process ordinarily is not of interest unless it contains an error message. The other logs have the same content as their counterparts for the object server child processes, above.

If you want to retain the log even when a Gem session process exits normally, edit the shell script `$GEMSTONE/sys/gemnetobject`. (For remote page servers, edit the shell script `$GEMSTONE/sys/runpgsvr` instead.) Make sure that the following line is *not* commented out:

```
unset GEMSTONE_CHILD_LOG
```

CAUTION

Any changes that you make to these scripts will apply to all other Gem sessions or remote page server processes, respectively.

NetLDI Logs

Each NetLDI creates a log file (*netLdiName.log*) in `/opt/gemstone/log` on the node on which it runs. (For compatibility with previous GemStone/S products, these directories can be in `/usr`.) This location and name can be overridden by including the option `-logname` when starting the NetLDI. Each NetLDI you start with the same name appends to one log, so it's a good idea to remove outdated messages occasionally.

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 startup command. The following log description for the default configuration may be helpful for comparison:

```
System password authorization is permitted.  
Authentication is required only to create processes.  
Process creation is permitted through user's HOME directory.  
Created processes belong to client's account.
```

The preceding lines map to NetLDI options in this way:

- Line 1 Authentication is not restricted.
- Line 2 Guest mode is not in use (`-g`), but authentication is not required for all NetLDI services (`-s`).
- Line 3 Services are not restricted to those listed in `$/GEMSTONE/bin/services.dat` (`-n`).
- Line 4 Captive accounts are not in use (`-aname`).

In some cases it is helpful to log additional information by starting the NetLDI in debug mode (`startnetldi -d`). The debug log records each exchange between the NetLDI and a client. Because the log becomes much larger, you probably won't want to use this mode routinely.

8.2 How to Audit the Repository

This section describes two levels of checks that you can perform on the repository.

- A *page audit* (page 8-10) typically is invoked only to ensure page-level consistency 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* (page 8-12) checks the consistency of the repository at the object level and generates useful statistics in the process. An object audit can be performed as part of routine maintenance and is always performed while GemStone is running.

To Perform a Page Audit

Page audits allow you to diagnose problems in the system repository by checking for consistency at the page level. You do not need to run this utility as part of routine maintenance of the repository.

The **pageaudit** utility can be run only on a repository that is not in use.

Page audits scan the rootpages in a repository, along with those pages used in the bitmap structures referenced by the rootpage. Many pages, including data pages, are not actually checked during a page audit. To check the integrity of all repository pages, perform a page audit, then perform an object audit as described on page 8-12.

To check for page-level problems, run **pageaudit** on the repository defined in your ordinary GemStone configuration by issuing this command at the operating system level:

```
% pageaudit [gemStoneName] [-zsystemConfig] [-eexeConfig] [-h]
```

where:

- *gemStoneName* is the name of the GemStone repository monitor.
- *systemConfig* is the system configuration file (page A-2).
- *exeConfig* is the executable configuration file (page A-5).

All of these arguments are optional in a standard GemStone configuration. If these options are not supplied, **pageaudit** uses `gs64stone` for *gemStoneName*.

- For more about the **pageaudit** command and its options, see page B-9.
- For online documentation, type **pageaudit -h** or **man pageaudit**.

As **pageaudit** runs, it prints repository statistics to the screen. For example:

```

PAGE AUDIT STATISTICS paris sun4u (Solaris 2.9
Generic_117171-08) - 03/08/07 08:55:16 PST
16384 bytes = 1 GemStone Page
1048576 bytes = 1 Mbytes
Repository Size                               53 Mbytes
Data Pages                                     6 Mbytes
Meta Information Pages                         1 Mbytes
Shadow Pages                                  0 Mbytes
Free Space in Repository                       44 Mbytes
**** Number of differences found in page allocation = 0.

```

Page Audit of Repository completed successfully.

The report contains the following statistics:

Repository Size	The total physical size of the repository; this is the same size that the operating system reports for an extent file.
Data Pages	This includes all pages referenced from the object table.
Meta Information Pages	Pages that contain only internal information about the repository, such as the object table.
Shadow Pages	Pages scheduled for scavenging by the reclaim task.
Free Space in Repository	Free space in the repository is computed as the number of free pages times the size of a page (16 KB). That value reflects the number of pages available for allocation to Gem session processes. It excludes space fragments on partially filled data 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 a Smalltalk Full Backup” on page 9-14.)

To Perform an Object Audit and Repair

Privileges required: `GarbageCollection`.

Object audits check the consistency of the repository at the object level. The output includes a description of any errors found and, depending on the particular method invoked, statistics about the Repository.

GemStone provides several choices about how the audit is conducted:

- The level of consistency checking required
- The minimum object size for which statistics are generated
- Whether the audit is optimized for speed on large repositories

All of these approaches abort the current transaction, and garbage collection and other administrative sessions are shut down for their duration.

To have the highest degree of confidence in the audit results, perform the object audit in single-user mode and perform a `reclaimAll` (the default). Logins are disabled for the duration of the audit, page reclamation is forced to complete, and additional consistency checks are made. If these conditions are not met, an appropriate error is returned immediately. The basic method is

```
Repository>>auditWithLimit: sizeLimit reclaimAll: reclaimArg
```

where *sizeLimit* is the object size threshold (bytes or Oops) below which statistics are not reported, and *reclaimArg* is a Boolean indicating whether an error should be returned if the conditions for a detailed consistency check are not met. The *sizeLimit* does not affect which objects are audited; it only affects reporting.

These convenience methods are provided:

```
Repository>>objectAudit
```

`objectAudit` is the simplest method to use. It reports all errors it encounters, but statistics are reported only for objects larger than 100000 bytes or Oops. If the system is in single-user mode and reclamation can be completed, detailed checks are performed. If these conditions are not met, the method silently performs more general checks.

```
Same as auditWithLimit: 100000 reclaimAll: true
```

```
Repository>>auditWithLimit: sizeLimit
```


`auditWithLimit`: lets you specify the reporting threshold for statistics. Like `objectAudit`, it performs detailed checks if the necessary conditions are met, or silently performs more general checks.

Same as `auditWithLimit`: *sizeLimit* `reclaimAll`: `true`

The above methods attempt to perform the `reclaimAll` function (if the system is in single-user mode), then begin with an optimized scan of the Object Table and the data pages. The audit results and object statistics are written to standard output. If you want to save the statistics, use `output push` within Topaz to redirect output to a log file. For information about the report, see “Understanding Object Audit Statistics” on page 8-16. If errors are detected, GemStone ordinarily re-scans the repository to provide detailed information.

Object Audits Without Reclaim

When a detailed check is not possible — for example, if there are other user sessions logged in — you can perform the object audit under less stringent conditions. However, the degree of confidence in the results is reduced because less checking is possible. The following methods are available for such conditions:

```
Repository>>objectAuditNoReclaim
Repository>>auditWithLimit: sizeLimit reclaimAll: false
```

These methods are the same as `objectAudit` and `auditWithLimit`:, respectively, but do not attempt to complete reclaiming of all shadowed and dead objects before the audit. (That is, `reclaimAll`: `false`.)

Quick Audits

The optimized method `Repository>>quickObjectAuditWithLevel: anInt` is intended for use with large repositories, where it runs substantially faster than `objectAudit` and `auditWithLimit`:. However, this method must be run in single-user mode, and the Admin GcGem must have had time to complete dead object finalization following any garbage collection activity. Object statistics are not reported in the interest of attaining the fastest performance.

`quickObjectAuditWithLevel: 1` is optimized to find the most common types of errors. The object table is not audited by this method, but most other checks done in the standard object audit (`Repository>>auditWithLimit`:) are performed. References to any non-existent, free or dead oops are reported as errors.

`quickObjectAuditWithLevel: 2` performs the same integrity audits as a level-1 quick audit. In addition, all object table entries are audited to verify the disk

address of each object. On large repositories, this method takes longer to complete than a level-1 quick audit.

Performing the Object Audit

To perform an object audit:

Step 1. Log in to GemStone using linked Topaz (**topaz -l**).

Step 2. Ensure that the system is in single-user mode:

```
topaz 1> printit
System stopUserSessions
%
```

For each active session (other than the one invoking it), this method aborts the transaction and terminates the session. It also suspends further logins. If you prefer, you can use `stopSession: aSession` to stop individual sessions by number.

NOTE

If the Gem is responsive, it usually terminates within milliseconds of either `stopUserSessions` or `stopSession:`. However, if a Gem is not active (for example, sleeping or waiting on I/O), the stone waits for `STN_GEM_TIMEOUT` seconds before forcibly stopping the session. If `STN_GEM_TIMEOUT` is set to 0, it will wait for 5 minutes.

Step 3. Start the Admin GcGem:

```
topaz 1> printit
System startAdminGcSession
%
```

Step 4. Start all Reclaim GcGems for which your repository is configured:

```
topaz 1> printit
System startReclaimGcSessions
%
```

Step 5. Send one of the audit messages to the repository. For example:

```
topaz 1> printit
SystemRepository objectAudit
%
```

Errors During the Object Audit

If errors are reported during the object audit, do the following to determine if there is a problem:

- Make sure you are the only user logged in (other than GcUser).
- Make sure that the Admin GcGem and any Reclaim GcGems are running to complete processing of dead objects.
- Run `markForCollection` (see page 10-26):

```
topaz 1> printit
SystemRepository markForCollection
%
```

(Alternatively, you can run the FDC/MGC process, as described on page 10-31.)

- Wait for the reclaim process (page 10-35) to complete.
- Repeat the object audit.

The audit involves a number of checks and specific error messages. The following categories illustrate their nature:

- 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). Errors of this type prevent a rescan for details.
- 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. The exceptions should be dead objects in the process of being reclaimed.
- Reclaiming — If the audit is being performed in single-user mode, reclamation should have removed all shadowed objects, which are the previous values of committed objects.

Error Recovery

If the errors are a few invalid object references, you may choose to repair them yourself. Use the Topaz object identity specification format `@identifier` to substitute nil or an appropriate reference for an invalid reference. For example, to replace an invalid reference in an instance of Array:

```
topaz 1> send @119873 at: 3 put: nil
```

You can have GemStone attempt appropriate repairs during the re-scan by invoking `Repository>>repairWithLimit:`. The following repairs illustrate their nature:

- `OopsNil` 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.

A descriptive message is displayed for each repair.

To have GemStone make the repairs, do the following:

Step 1. Log in to GemStone using linked Topaz (**topaz -l**).

Step 2. Make sure you are the only user logged in, other than administrative sessions. (See page 8-14, Step 2.) The next step will stop the `GcUser` session and disable logins for its duration.

Step 3. Send the message `repairWithLimit:sizeLimit` to the repository, specifying an appropriate threshold for reporting object statistics. For example, to use the same reporting limit as `objectAudit`:

```
topaz 1> printit
SystemRepository repairWithLimit: 100000
%
```

Because `repairWithLimit:` includes an object audit, some administrators prefer to use this method initially rather than repeating the audit in the process of repairing errors found by a previous audit. However, `repairWithLimit:` requires that you be the only user logged in.

Understanding Object Audit Statistics

Figure 8.1 shows a representative set of statistics resulting from an object audit. The report is in three parts:

1. A list of all objects (including private ones) that exceed a certain size limit, which in this example is 5000 bytes or `Oops`. The method `objectAudit` has a preset limit of 100000 as the smallest object to be included in the list.

Inspect this list for large objects created by your application.

2. Statistics about invisible (private) classes that are reserved for GemStone's use. The number of these classes varies from release to release, and some may not be used in a particular release.
3. Statistics about instances of visible classes, including instances within the kernel.

Of particular interest are the number of objects (which you can compare with the number reported by an audit of the initial GemStone repository) and the average size of an object's value. The size statistic may be helpful in estimating the eventual size of your repository (see "Estimating Extent Size" on page 1-19). In this example, the objects occupy an average of 75.9 bytes each plus an overhead of 26 bytes each.

Object tags are hidden instance variable slots in all objects except SmallIntegers, Booleans, and UndefinedObjects (nils). For further information, see the comment for `Object>>tagAt:` and `tagAt:put:` in the image.

Figure 8.1 Statistics from an Object Audit

```

topaz 1> printit
SystemRepository auditWithLimit: 5000
%
--reclaimAll: changed StnSignalAbortCrBacklog from 20 to 3
--setGcConfig: set reclaimMinPages to 1
--reclaimAll: changed reclaimMinPages from 40 to 1
--reclaimAll: using reclaimDeadEnabled true
Object audit as of 03/05/07 11:39:05 PST
Summary of objects whose sizes exceed 5000 Bytes or Oops:

  ObjectID      Class  ClassName      LogicalSize
-----
    105809      105881 WidgetCollection      22340 Oops
      49007         523 Array                6011 Oops
      82621         585 String                6387 Bytes
      32025         585 String                5747 Bytes
    149093      105881 WidgetCollection      9113 Oops
      40977         585 String                6029 Bytes
      90861         585 String                8122 Bytes
  ...

----- Object Statistics Summary -----

----- Instances of invisible (private) classes -----
Number of instances:          22
      Total size:             155 K Bytes
      Average size:           7.0 Bytes

      Class: 1635 Instances:          0 Total Size:          0 K Bytes
      Class: 1636 Instances:          0 Total Size:          0 K Bytes
      Class: 1637 Instances:         22 Total Size:         155 K Bytes
      Class: 1638 Instances:          0 Total Size:          0 K Bytes
      Class: 1639 Instances:          0 Total Size:          0 K Bytes
      Class: 1640 Instances:          0 Total Size:          0 K Bytes
      Class: 1641 Instances:          0 Total Size:          0 K Bytes
  ...

----- Instances of visible classes -----
Number of objects      :      72081
Total Size             :      6781 K Bytes
size of Object Headers :      1407 K Bytes
size of Object Values  :      5344 K Bytes
size of Object Tags    :           0 K Bytes
average of Object Value:       75.9 Bytes

Object Audit: Audit successfully completed; no errors were detected.
--setGcConfig: set reclaimMinPages to 40
--postReclaimAll: restored reclaimMinPages to 40
true
topaz 1>

```

Larger Application Objects

Private Classes Reserved for Gem and Stone

Instance Statistics

8.3 Monitoring Performance

As part of your ongoing responsibilities, you may find it useful to monitor performance of the object server or individual session processes. You can obtain information separately about page reads and writes, or you can obtain more detailed cache statistics about that session.

This section discusses two ways in which GemStone supports monitoring:

- Command-line facilities to determine the general status.
- Smalltalk messages to access statistics maintained in the shared page cache.

In addition, GemStone provides two applications—*statmonitor* and VSD (Visual Statistics Display)—that allow you to monitor performance in depth. For details, see Appendix F.

To Monitor Page Reads and Writes by a Session

Privileges required: Statistics.

You can obtain information about session I/O by invoking the methods `System class>>pageReads` and `System class>>pageWrites` from that session. These methods return the number of reads and writes performed by that session since its start. For example:

```
topaz 1> printit
System pageReads
%
19
topaz 1> printit
System pageWrites
%
0
```

To Monitor Cache Statistics

As mentioned above, the utility programs *statmonitor* and VSD allow you to examine cache statistics, the latter in a graphical way. For details on their use, see Appendix F.

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 VSD.)

A process can only access statistics that are kept in the shared page cache to which it is attached. This means that processes that are remote from the Stone, and are thus attached to a local shared page cache, 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. To find the value of a particular statistic, you must first determine the process slot for that server process and the index of the statistic within the process slot. Based on this information, you can then locate the data of interest.

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, Page Manager, Admin and Reclaim GcGems, and Symbol Gems, 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.

You can use the method `System class >> myCacheProcessSlot` to return the process slot in the shared page cache that corresponds to the calling process.

To determine the index of a particular cache statistic, use the method `System Class >> cacheStatisticsDescription`. This method returns an array containing the names of all the statistics. The offsets given in this method may change with each release, so it is not a good idea to hard-code this, although you may wish to cache the descriptions. The statistic names won't change, so looking up via the name is safe.

You can use the method `System class >> cacheStatistics: aProcessSlot` to obtain an array of statistics for the process assigned to the specified slot.

For Gem processes, you can use the alternative methods `System class >> cacheStatisticsForSessionId:` or `System class >> cacheStatsForGemWithName:` to look up the statistics for a Gem process by `sessionId` or `name`, respectively. These methods return `nil` if the process with that `sessionId` or Gem name is not attached to the same cache as the calling process.

Each of these methods returns an array containing the current values for all statistics for the given process. Each statistics value is at the same offset as its name in the `cacheStatisticsDescription` array. Since not every process type records values for every statistic, the unused elements will have 0 value at that offset.

For example, to find out the index for the PossibleDeadKobjs, you could use this code:

```
topaz 1> printit
System cacheStatisticsDescription indexOf:
'PossibleDeadKobjs'.
%
113
```

The following code illustrates one way to look up the value for the Stone's PossibleDeadKobjs:

```
topaz 1> printit
| index |
index := System cacheStatisticsDescription indexOf:
'PossibleDeadKobjs'.
(System cacheStatistics: 1 "Stone" ) at: index.
%
0
```

This example shows how you might go about finding the calling session's PageReads:

```
topaz 1> printit
| index |
index := System cacheStatisticsDescription indexOf:
'PageReads'.
(System cacheStatistics: System myCacheProcessSlot) at:
index.
%
966
```

To make it easier for you to track cache statistics for specific Gems, you can explicitly give each Gem a unique name. The private 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. Note that `_cacheName:` is a private method; as such, it is provided here only for your convenience, and is subject to change in future releases.

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 `SessionStat0...SessionStat47`.

You can use the following methods to read and write the session cache statistics:

System Class >> `_sessionCacheStatAt: aIndex`

Returns the value of the statistic at the designated index (must be in the range 0..47).

System Class >> `_sessionCacheStatAt: aIndex put: aValue`

Assigns a value to the statistic at the designated index (must be in the range 0..47) and returns the new value.

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 `GlobalStat0...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.

Cache Statistics

This section lists the GemStone/S 64 Bit statistics in alphabetical order. The heading indicates the processes for which the statistic is applicable: Stone, Gem, SPC (shared page cache) monitor, Pgsvr (AIO page server), or all.

NOTE

Certain statistics not listed in this chapter, but visible in VSD displays, are for internal purposes only.

AbortCount (Gem)

The number of aborts executed by a Gem process (or by an application linked to a Gem) since the Gem was most recently started.

AioCkptCount (Pgsvr)

The number of dirty pages written from the shared page cache to disk to satisfy a checkpoint.

AioDirtyCount (Pgsvr)

The number of dirty pages written from the shared page cache to disk by Stone's AIO page server during normal operation.

AioNumBuffers (Stone)

The current setting of STN_MAX_AIO_REQUESTS

AioNumEmptyBuffers (Stone)

The number of internal AIO buffers that are currently empty. The stone will block when starting a tranlog write unless at least 3 buffers are empty.

AioRateLimit (Pgsvr)

The current I/O rate being used by the page server, expressed in I/O operations per second. The page server will perform no more than this number of I/O operations per second on average.

AioRateMax (Pgsvr)

The current maximum I/O rate being used by the page server, expressed in I/O operations per second. The page server will perform no more than this number of I/O operations per second on average.

AsyncFlushesInProgress (Pgsvr)

The number of pending extent flush operations.

BackupRecordPagesWrittenByGem (SPC monitor)**BackupRecordPagesWrittenByStone (SPC monitor)**

The number of backup record pages in the cache because of a write done by a Gem or by the Stone, respectively.

BitlistPagesWrittenByGem (SPC monitor)**BitlistPagesWrittenByStone (SPC monitor)**

The number of bitlist pages in the cache because of a write done by a Gem or by the Stone, respectively. A bitlist page is a form of a bit array.

BitmapPageReads (all)

The number of bitmap pages read by the process since it was last started. These page reads are actual disk reads and not reads from the shared page cache.

BmCHeapPages (Gem, Stone)

The number of temporary bitmap pages allocated on the C heap of the process.

BmInternalPagesWrittenByGem (SPC monitor)**BmInternalPagesWrittenByStone (SPC monitor)**

The number of bitmap internal pages in the cache because of a write done by a Gem or by the Stone, respectively.

BmLeafPagesWrittenByGem (SPC monitor)
BmLeafPagesWrittenByStone (SPC monitor)

The number of bitmap leaf pages in the cache because of a write done by a Gem or by the Stone, respectively.

CHeapSizeKB (all)

The size (in KB) of the C heap area; that is, the total size of results of `UtlMalloc` for which `UtlFree` has not been called. This statistic is computed only in slow or `fastdebug` executables.

CheckpointCount (Stone)

The number of checkpoints that have been written since the Stone repository monitor was last started. Writing a checkpoint implies that all of the data and meta information needed to recover the data corresponding to the commit record associated with the checkpoint have been written to the disk(s) containing the extent(s) that make up the repository. Thus, the last checkpoint in the transaction log determines how much data in the log must be recovered when there is a system crash.

In full logging mode, the checkpoints are controlled completely by the `STN_CHECKPOINT_INTERVAL` configuration parameter (page A-25). In partial logging mode, a checkpoint may be written more often if the size of the transaction exceeds the value set by the configuration parameter `STN_TRAN_LOG_LIMIT` (page A-36). If partial logging is in use, a rapidly increasing `CheckpointCount` indicates that `STN_TRAN_LOG_LIMIT` may be set too small.

CheckpointState (Stone)

This internal statistic indicates the state of the checkpoint process:

- 0 — checkpoint not active
- 1 — AIO servers writing pages
- 2 — pre-fsync processing
- 3 — synchronous fsync
- 4 — asynchronous fsync
- 5 — second fsync
- 6 — dispose alloc pages shadowed
- 7 — write log record

These values are provided for information only and are subject to change without notice.

ClassesRead (Gem)

The number of classes copied into VM memory since the start of session.

ClientLostOtsSent (Pgsvr)

The number of Lost OT root signals sent from the stone to the page server's client.

ClientPageReads (Pgsvr)

The number of pages that have been transmitted by a page server to its client. This statistic is implemented only for cache slots used by a page server.

ClientPageWrites (Pgsvr)

The number of pages that have been transmitted by a client to its page server. This statistic is implemented only for cache slots used by a page server.

ClientPid (Pgsvr)

The process ID of the client process associated with this AIO page server process.

ClientSigAbortsSent (Pgsvr)

The number of SigAborts sent from the stone to the page server's client.

ClockHandFrameId (Pgsvr)

The shared page cache frame ID that the page server clock hand currently references.

CodeCacheSizeBytes (Gem)

The total size in bytes of copies of GsMethods that are in the code generation area and ready for execution, as of the end of mark/sweep.

CodeGenGcCount (Gem)

The number of times the code generation area has been garbage collected.

CommitCount (Gem)

The number of commits executed by a Gem process (or application linked to a Gem) since the Gem was most recently started.

CommitQueueAddedToRunQueueCount (Stone)

The number of times sessions from the commit queue were added to the run queue.

CommitQueueAddedToRunQueueSessionCount (Stone)

The number of sessions from the commit queue which were added to the run queue.

CommitQueueSessionNotReadyCount (Stone)

The number of times the stone was ready to assign the commit token to a session in the commit queue but no session was not ready to receive the token.

CommitQueueSize (Stone)

The number of Gem session processes waiting for the commit token.

CommitQueueThreshold (Stone)

The current setting of the STN_COMMIT_QUEUE_THRESHOLD configuration parameter. This setting determines how large the commit queue must be before the stone will defer commit record disposal. A value of -1 indicates this feature is disabled and commit record disposal will never be deferred.

CommitRecordCount (Stone)

The number of outstanding commit records that are currently being maintained by the system. A number larger than the STN_SIGNAL_ABORT_CR_BACKLOG configuration option (page A-35) indicates that there is a process in a transaction that is preventing the Stone from reclaiming (garbage collecting) the resources associated with those commit records. Large values are usually accompanied by continuing growth in the size of the repository.

CommitRecordDisposalsDeferredCount (Stone)

The number of times the stone deferred commit record disposal because the number of sessions in the commit queue exceeded the STN_COMMIT_QUEUE_THRESHOLD setting.

CommitRecordDisposalState (Stone)

An internal statistic used to analyze the commit record disposal routines.

CommitRecordPagesWrittenByGem (SPC monitor)
CommitRecordPagesWrittenByStone (SPC monitor)

The number of commit record pages in the cache because of a write done by a Gem or by the Stone, respectively.

CommitRecordsDisposable (Stone)

The number of commit records queued for disposal. These are no longer being referenced by any session.

CommitRecordsDisposedCount (Stone)

The total number of commit records disposed by the Stone.

CommitRecordsReadAbortCount (Gem)

The number of commit records read while processing an abort.

CommitRecordsReadCommitWithoutTokenCount (Gem)

The number of commit records read while processing a commit and the Gem is waiting for the commit token.

CommitRecordsReadCommitWithTokenCount (Gem)

The number of commit records read while processing a commit and the Gem has the commit token.

CommitRetryFailureCount (Gem)

The number of commits that failed after exceeding the retry count.

CommitsSinceLastEpoch (Stone)

Number of commits, excluding reclaim commits by GcGems, since the start of the currently open epoch.

CommitTokenSession (Stone)

The session ID of the gem holding the commit token. If no gem is holding the commit token, the value will be zero.

CountBagInteriorPagesWrittenByGem (SPC monitor)
CountBagInteriorPagesWrittenByStone (SPC monitor)

The number of count bag interior pages in the cache because of a write done by a Gem or by the Stone, respectively.

CountBagLeafPagesWrittenByGem (SPC monitor)
CountBagLeafPagesWrittenByStone (SPC monitor)

The number of count bag leaf pages in the cache because of a write done by a Gem or by the Stone, respectively.

CountMapLeafPagesWrittenByGem (SPC monitor)
CountMapLeafPagesWrittenByStone (SPC monitor)

These statistics are obsolete; the page kind is no longer used.

DataPageReads (all)

The number of data pages read by the process since it was last started. These page reads are actual disk reads and not reads from the shared page cache.

DataPagesWrittenByGem (SPC monitor)
DataPagesWrittenByStone (SPC monitor)

The number of data pages in the cache because of a write done by a Gem or by the Stone, respectively.

DeadDataPagesWrittenByGem (SPC monitor)
DeadDataPagesWrittenByStone (SPC monitor)

The number of dead data pages in the cache because of a write done by a Gem or by the Stone, respectively. Dead data pages are intended for disposal.

DeadNotReclaimedKobj (Stone)

The number of objects that have been determined to be dead (current sessions have indicated they do not have a reference to these objects) but have not yet been reclaimed. (Units: 1000s of objects.)

DeadObjsReclaimedCount (Stone)

The total number of dead objects reclaimed since the Stone repository monitor process was last started. For a system in “steady state” for a particular application, look for a uniform discovery rate per garbage collection epoch. Increasing the

duration of the epoch should increase this value, but that could also cause larger swings in the amount of free space in the repository.

DeferCkptCompleteCount (Stone)

The number of pending commit operations for which the checkpoint will allow itself to be deferred before it completes.

DepMapKeysChanged (Gem)

The total number of objects with DependencyMap changes committed by this session.

DirtyListSize (Gem)

The number of modified committed objects in the temporary object memory dirty list.

DirtyPageSweepCount (Pgsvr)

The number of times the page server has swept the cache for dirty pages or frames to add to the free list.

EmptyBmLeafPagesWrittenByGem (SPC monitor)

The number of empty bitmap leaf pages in the cache because of a write done by a Gem.

EmptyBmLeafPagesWrittenByStone (SPC monitor)

The number of empty bitmap leaf pages in the cache because of a write done by the Stone.

EpochForceGc (Stone)

This statistic is 1 when a user has requested that Epoch garbage collection be started manually, and is otherwise zero.

EpochGcCount (Stone)

The number of times that the Epoch garbage collection process was run by the GcGem since the GcGem was started. For a system in steady state, look for uniform periods between runs or a uniform run rate.

EpochNewKobjs (Stone)

The number of new objects that were created during the last epoch. (Units: 1000s of objects.)

EpochPossibleDeadKobjs (Stone)

The number of possible dead objects found by the last Epoch garbage collection. (Units: 1000s of objects.)

EpochScannedKobjs (Stone)

The number of objects scanned by the last Epoch garbage collection. (Units: 1000s of objects.)

ExportedSetSize (Gem)

The number of objects in the ExportSet. The ExportSet is a collection of objects for which the Gem process has handed out an Oop to GemBuilder or Topaz. Objects in the ExportSet are prevented from being automatically garbage collected in the Gem's temporary object memory (see page 10-11). The ExportSet is used to guarantee referential integrity for objects only referenced by an application, that is, objects that have no references to them within the Gem.

The application program is responsible for timely removal of objects from the ExportSet. The contents of the ExportSet can be examined using hidden set methods defined in class System. In general, the smaller the size of the ExportSet, the better the performance is likely to be. There are a couple of reasons for this relationship. The ExportSet is one of the root sets used for garbage collection. The larger the ExportSet, the more likely it is that objects that would otherwise be considered garbage are being retained. One threshold for performance is when the size of the export set exceeds 2K objects. When its size is smaller than 2K objects, the export set is stored as a single disk page. When its size is larger than 2K, the export set occupies more than one page and is likely to cause additional I/O.

ExtentFlushCount (all)

The cumulative number of file flush operations performed on any extent by the process. Note that extents residing on raw partitions do not require flushing. On UNIX systems, file flushes are performed by calling the `fsync()` function. During a checkpoint, each extent is flushed once, except for the primary extent which is flushed twice. Most extent flushes are performed by the AIO page servers.

FailedCommitCount (Gem)

The number of attempts to commit that failed due to concurrency conflicts.

FragmentBmPagesWrittenByGem (SPC monitor)
FragmentBmPagesWrittenByStone (SPC monitor)

The number of bitmap fragment pages in the cache because of a write done by a Gem or by the Stone, respectively.

FrameCount (SPC monitor)

The size of the shared cache in frames.

FramesAddedToFreeList (all)

The number of frames added to the free list since the session (for Gems), shared page cache, or Stone started.

FramesFromFreeList (all)

The number of times the process acquired a page frame in the shared page cache from the list of free frames.

FramesFromFindFree (all)

The number of times the process acquired a page frame in the shared page cache by scanning the cache entries. The process tries to find free frames this way instead of taking them from the free list when the number free is below the value set by the `GEM_FREE_FRAME_LIMIT` configuration option (page A-14). While scanning for free frames under those conditions is desirable from a system perspective, it represents additional overhead for the particular session.

FreeFrameCacheNumFrames (all)

The number of frames present in the free frame cache for the process.

FreeFrameCacheSize (all)

The number of frames that the process will add or remove from the shared free frame list in a single operation. A value of zero indicates that free frames are not cached and will be added or removed from the shared free frame list one at a time.

FreeFrameCount (SPC monitor)

The number of unused page frames in the shared page cache. This statistic gives some indication of the utilization of the cache, but it is not tunable. This statistic is valid (non-zero) only for the shared page cache monitor process slot.

FreeFrameLimit (all)

When the number of free frames in the shared page cache is less than the FreeFrameLimit, the Gem scans the cache for a free frame rather than use one from the free frame list, so that the Stone process can use the remaining free frames.

FreeOopsK (Stone)

The number of free OOPs in the free list that have not been allocated to a Gem *and* committed. (Units: 1000s of OOPs.)

FreePages (Stone)

The size of the free page pool for the repository. Free space in the repository is calculated at 16 KB for each page in the free pool.

GciRpcCommandsServiced (Gem)

The number of GCI RPC commands serviced by this Gem.

GcVoteState (Stone)

GcVoteState tracks the progress of a number of phases within garbage collection. Possible values are:

- 0 – Not voting
- 1 – Gems are voting on the possible dead set
- 2 – Vote completed, awaiting start of Possible Dead Write-Set Union Sweep (PDWSUS).
- 3 – PDWSUS in progress
- 4 – PDWSUS completed

Note that all of these phases must complete before the PossibleDead objects can be “promoted” to DeadNotReclaimed objects. The Admin GcGem is responsible for performing the possible dead write set union sweep, and must be running for this to occur.

For details about the voting phase of garbage collection, see “What Happens to Garbage?” on page 10-6.

GcWsUnionSweepCount (Stone)

The total number of sweeps of the possible dead write set union that have been done by the Admin GcGem since it started.

GemFreePages (Gem)

The number of free pages that the VM or Gem has acquired but has not yet used.

GemHasCommitToken (Gem)

A boolean that indicates whether the Gem holds the commit token.

GemsInCacheCount (SPC monitor)

The total number of Gems using the shared page cache whose process slot you are viewing. This is useful for distinguishing Gems using a local shared page cache from those using a remote shared page cache.

GlobalDirtyPageCount (SPC monitor)

The total number of pages in the shared cache that are dirty but not yet eligible for asynchronous writing to the disk because they have not yet been committed. If this value is very large, then very large transactions may be filling the cache.

Otherwise, if the Stone repository monitor is running on this cache, the Stone's private page cache size may be too small. This statistic is available only for the shared page cache monitor's slot (currently Slot 0).

GlobalStat0 — GlobalStat47 (Stone)

User-defined statistics that can be written and read by any Gem on any Gem server; that is, these statistics are stored in the shared page cache of the Stone, rather than that of the machine on which the Gem is running. There are 48 global cache statistic slots available. For details, see "Global Session Statistics" on page 8-22.

GsMsgCount (Stone)

The number of messages processed by the Stone on behalf of Gems. This statistic can help determine how busy the Stone is.

GsMsgKind (Stone)

An integer identifying the kind of message the Stone is currently processing.

GsMsgSessionId (Stone)

The session identifier of the Gem for which the Stone is currently processing a message.

InvalidPagesWrittenByGem (SPC monitor)
InvalidPagesWrittenByStone (SPC monitor)

The number of invalid (i.e. empty) pages in the cache because of a write done by a Gem or by the Stone, respectively.

LastSessionFatalError (Stone)

The session ID of the last session that reported a fatal error to the stone.

LastSessionIdStopped (Stone)

The session ID of the last session that was sent a stop session message.

LastSessionIdTerminated (Stone)

The session ID of the last session that was forcibly logged off by the stone.

LastSessionLostOt (Stone)

The session ID of the last session that was sent a SigLostOtRoot signal.

LastSessionSigAbort (Stone)

The session ID of the last session that was sent a SigAbort.

LastWakeupInterval (SPC monitor)

The average number of milliseconds that the shared page cache monitor is pausing between recalculations. If this value is low, the monitor is relatively busy; if high (for example, greater than 500 ms), the monitor is relatively quiet.

LdiThreadOperations (Stone)

The number of wakeups from semaphore wait in stone NetLdiCall Thread.

LocalCacheAllocatedPceCount (Gem, Stone)

The number of page cache entries that the process has allocated for collision chains.

LocalCacheFreeFrameCount (Gem, Stone)

The number of frames in the private page cache that are free.

LocalCacheFreePceCount (Gem, Stone)

The number of page cache entries in the free pool.

LocalCacheOverflowCount (Gem, Stone)

The number of times that a page was moved from private cache to the shared cache because the private cache was full.

LocalCachePceCountLimit (Gem, Stone)

The maximum number of page cache entries that the process will allocate before performing a reclaim.

LocalCachePceReclaimCount (Gem, Stone)

The number of page cache entry reclaim operations performed by the process.

LocalCacheStalePcesRemovedCount (Gem, Stone)

The number of stale page cache entries that the process has removed from its private page cache lookup table. Stale PCEs occur when a reference to page in the shared cache becomes invalid because the page is removed from the cache by another process.

LocalCacheValidPcesRemovedCount (Gem, Stone)

The number of valid page cache entries removed by a reclaim operation.

LocalDirtyPageCount (SPC monitor)

The total number of pages in the shared cache that are dirty and eligible for asynchronous writing to the disk. The Stone's AIO page server will write these pages to the disk. This statistic is available only for the shared page cache monitor's slot (currently Slot 0).

LocalPageCacheHits (all)

The number of times a page lookup found the page in either the private or shared page cache. No I/O was required to access the page.

LocalPageCacheMisses (all)

The number of times a page was not found in either the private or shared cache and a read operation was required to get the page.

LocalPageCacheWrites (all)

The number of times a page had to be written. With the shared page cache enabled, this statistic counts the writes from the private cache to the shared cache. If the shared page cache is disabled, it counts the pages written to disk.

Lock1WaitQueueSize (Stone)

The number of sessions waiting for the Application 1 object lock. (System `waitForApplicationWriteLock:queue:autoRelease:`)

Lock2WaitQueueSize (Stone)

The number of sessions waiting for the Application 2 object lock. (System `waitForApplicationWriteLock:queue:autoRelease:`)"

LockReqQueueSize (Stone)

The number of Gem session processes waiting for a commit to complete so that their lock request can be serviced.

LogHighQueueAdds (Stone)

Number of additions to the LogHighQueue. See LogHighQueueSize.

LogHighQueueSize (Stone)

The number of sessions waiting for transaction log writing to be not saturated. This queue holds mostly committing sessions

LoginQueueSize (Stone)

The number of sessions waiting for login completion.

LoginRequestsCount (Stone)

The total number of login requests processed since the Stone started.

LogLowQueueAdds (Stone)

Number of additions to the LogLowQueue. See LogLowQueueSize.

LogLowQueueSize (Stone)

The number of lower priority sessions waiting for transaction log writing to be not saturated. This queue holds primarily sessions logging resourceIds such as `possibleDead` or `notDead`.

LogRecordPagesWrittenByGem (SPC monitor)
LogRecordPagesWrittenByStone (SPC monitor)

The number of transaction log record pages in the cache because of a write done by a Gem or by the Stone, respectively.

LogRecordsIoCount (Stone)

The number of physical write operations performed on the transaction logs since the Stone repository monitor process was last started. The minimum write to a transaction log is 512 bytes (one log record). The maximum number of bytes written in a single I/O to the transaction log is 64K. The implication for performance tuning is that to achieve the best throughput (in transactions per second) you would like to have as few as possible writes to the transaction logs. The technique for achieving this is to tune the size of the transactions so that each transaction writes one or more completely filled 64K records.

LogRecordsWritten (Stone)

The number of log records that have been written to the transaction logs since the Stone repository monitor process was last started. The size of a log record is 512 bytes.

LogWaitQueueSize (Stone)

The size of the queue that holds sessions waiting for space to become available in a transaction log. This queue should be empty or nearly so unless the space for logging transactions has been exhausted.

LostOtPagesWrittenByGem (SPC monitor)
LostOtPagesWrittenByStone (SPC monitor)

The number of lost OT pages in the cache because of a write done by a Gem or by the Stone, respectively.

LostOtsReceived (Gem)

The number of Lost OT Root signals received and recognized by this session.

LostOtsSent (Gem)

The number of Lost OT Root signals the Stone has sent to this session, although the session may be in a sleep or I/O wait state and not yet aware of having received the signal. (See LostOtsReceived (Gem), above.)

MarkSweepCount (Gem)

The number of mark/sweeps executed by the in-memory garbage collector.

MaxVotingSessions (Gem)

Maximum number of sessions that can concurrently vote on possible dead objects at the end of an MFC (`markForCollection`) or Epoch garbage collection.

MeSpaceAllocatedBytes (Gem)

The number of bytes allocated for `remSet`, in-memory `oopMap`, and map entries.

MeSpaceUsedBytes (Gem)

The number of bytes occupied by `remSet`, in-memory `oopMap`, and in-use map entries.

MessageKindToStone (Gem)

The message type of the most recent message sent to the Stone.

MessagesToStnProcessingCommit (Gem)

The number of messages sent to the Stone while the gem is processing its part of the commit.

MessagesToStnStoneCommit (Gem)

The number of messages sent to the Stone while the Stone is processing its part of the commit.

MessagesToStnWaitingForCommit (Gem)

The number of messages sent to the Stone while waiting for the commit token.

MessagesToStone (Gem)

The number of messages sent by the Gem session process to the Stone repository monitor using shared memory as the channel.

MethodsRead (Gem)

The number of `GsMethods` copied into VM memory since the start of this session.

MilliSecPerIoSample (all)

MilliSecPerIoSample is used as a parameter to implement the configurable I/O limit for GemStone processes. Because a process's I/O rate currently is sampled every five I/O operations, this statistic is computed as $1000 / (\text{ioLimit} * 5)$. A value of 1 means that the process has no I/O limit. If the time in milliseconds since the last sample equals or exceeds MilliSecPerIoSample, the process can perform another I/O operation; if not, the process sleeps. MilliSecPerIoSample is particularly useful in limiting I/O rate of a process that is executing a long-running operation. If you want to calculate the current I/O limit, it is given by $1000 / (\text{MilliSecPerIoSample} * 5)$.

MultObjPagesWrittenByGem (SPC monitor)**MultObjPagesWrittenByStone (SPC monitor)**

The number of multiple object pages in the cache because of a write done by a Gem or by the Stone, respectively.

NetWriteThreadSocketWrites (Stone)

The number of socket write operations attempted in stone NetWrite thread.

NetWriteThreadWakeup (Stone)

The number of wakeups from semaphore wait in stone NetWrite thread.

NewGenSizeBytes (Gem)

The number of used bytes in the new generation at the end of mark/sweep.

NewObjsCommitted (Gem)

The number of new objects committed by the most recent transaction committed by this process.

NewSymbolRequests (Gem)

The number of symbol creation requests by a session to the symbol creation Gem.

NewSymbolsCount (Gem)

The number of new symbols created by this session.

NextSleepTime (Pgsvr)

The number of milliseconds that the page server will sleep during the next sleep period. This value is adjusted to regulate the IO rate of the page server.

NotifyQueueSize (Stone)

The number of Gem session processes using notifiers.

NumCacheWarmers (Stone)

The number of cache warmer gem processes currently operating on the shared cache.

NumInLdiQueue (Stone)

The number of uncompleted requests in the stone NetLdiCall thread input list.

NumInMainInpQueue (Stone)

The number of requests from other threads in the stone main thread input list.

NumInNetReadWorkList (Stone)

The number of tasks in the NetRead thread work list, produced by NetPoll action functions and not yet processed by the NetRead thread.

NumInNetWriteQueue (Stone)

The number of uncompleted requests in the input list to the stone NetWrite thread.

NumLiveSoftRefs (Gem)

The number of instances of SoftReference in temporary object memory found during an attempt to clear SoftReferences. This counter remains at zero until temporary object space grows to at least GEM_SOFTREF_CLEANUP_PERCENT_MEM (page A-18).

NumNonNilSoftRefs (Gem)

The number of instances of SoftReference in temporary object memory with non-nil, non-special values, that were found during an attempt to clear SoftReferences. This counter remains at zero until temporary object space grows to at least GEM_SOFTREF_CLEANUP_PERCENT_MEM (page A-18).

NumRefsStubbedMarkSweep (Gem)

The number of references contained in copies of committed objects that were stubbed (converted to a Pom objectId) by in-memory mark/sweep.

NumRefsStubbedScavenge (Gem)

The number of in-memory references that were stubbed (converted to a Pom objectId) by in-memory scavenge.

NumSharedCounters (SPC monitor)

The number of shared counters for this shared cache.

NumSoftRefsCleared (Gem)

The number of times that the in-memory garbage collector has cleared the `value` instance variable in instances of `SoftReference`.

NumVotingSessions (Stone)

The number of sessions that the stone has requested to vote on possible dead objects.

ObjectMemoryGrowCount (Gem)

On AIX and HPUX only, the number of times object memory was grown or shrunk by allocating a new virtual memory region with `mmap()`.

ObjectsRead (Gem)

The number of committed objects copied into VM memory since the start of this session.

ObjectsRefreshed (Gem)

The number of committed objects in VM memory that have been re-read from the page cache after transaction boundaries, since the start of this session.

ObjectTablePageReads (all)

The number of object table pages read by the process since it was last started. These page reads are actual disk reads and not reads from the shared page cache.

ObjsCommitted (Gem)

The number of objects committed by the most recent transaction committed by this process.

OldestCrSession (Stone)

The session ID of a session referencing the oldest commit record. Note that more than one session may reference a commit record. A value of -1 indicates that the oldest commit record is not referenced by any session.

OldestCrSessNotInTrans (Stone)

The session ID of a session that is not in a transaction that is currently referencing the oldest commit record, which may be preventing it from being reclaimed.

OldGenSizeBytes (Gem)

The number of used bytes in the old generation at the end of mark/sweep.

OmBytesFlushed (Gem)

The total number of temporary object memory bytes flushed to Pom during commit attempts for this session.

OopNumberKHighWaterMark (Stone)

The highest number of object identifiers allocated by the system. (Units: 1000s of objects.)

OopsReturnedByGemsCount (Stone)

The total number of free oops returned to the stone by any gem.

OtherPageReads (all)

The number of pages read by the process that were not object table, data, or bitmap pages since the process was started. These page reads are actual disk reads and not reads from the shared page cache.

OtInternalPagesWrittenByGem (SPC monitor)**OtInternalPagesWrittenByStone (SPC monitor)**

The number of object table internal pages in the cache because of a write done by a Gem or by the Stone, respectively.

OtLeafPagesWrittenByGem (SPC monitor)
OtLeafPagesWrittenByStone (SPC monitor)

The number of object table leaf pages in the cache because of a write done by a Gem or by the Stone, respectively.

PageDisposesDeferred (Stone)

The number of times a page disposal had to be deferred. This deferral can be caused by an asynchronous operation (checkpoint) being in progress on the page or by the page being attached or locked.

PageIoCount (all)

The number of page I/O (page read or page write) calls done by the process since it was last started. Each I/O call may read or write more than one page.

PageIoTimeOverallAvg (all)

The average duration of a page I/O (page read or page write) call in microseconds.

PageIoTime10SampleAvg (all)

The average duration of a page I/O (page read or page write) call in microseconds. The average is computed for the last ten I/O operations, and is updated every ten I/O operations.

PageIoTime100SampleAvg (all)

The average duration of an I/O call (page read or page write) in microseconds. The average is computed for the last 100 I/O operations, and is updated every ten I/O operations.

PageLocateCount (all)

The number of times that the process located a page. The page may have been read from disk or found in the cache.

PageMgrPagesNotRemovedFromCachesCount (Stone)

The total number of pages that the Page Manager was unable to remove from one or more shared page caches.

PageMgrPagesPendingRemovalRetryCount (Stone)

The current number of pages that could not be removed from shared page caches by the page manager on the first attempt and are waiting to be retried.

PageMgrPagesReceivedFromStoneCount (Stone)

The total number of pages that the Page Manager session received from the Stone to remove from shared page caches.

PageMgrPagesRemovedFromCachesCount (Stone)

The total number of pages that the Page Manager has successfully removed from all shared page caches.

PageMgrRemoveFromCachesCount (Stone)

The total number of times that the Page Manager has attempted to remove pages from shared page caches.

PageMgrRemoveFromCachesPageCount (Stone)

The total number of pages that the Page Manager has attempted to remove from shared page caches. This statistic includes pages processed by page removal retry operations, which occur whenever a page cannot be removed from a shared page cache on the first attempt.

PageMgrRemovePagesFromCachesPollCount (Stone)

The number of times that the Page Manager called poll() or select() to determine which cache page servers have completed removing pages from their shared caches. This statistic represents the value during the most recent page disposal operation and is not cumulative. It varies between zero (when there are no remote shared caches on the system) and the number of remote shared page caches.

PageMgrTimeWaitingForCachePgsvrs (Stone)

The total amount of real time in milliseconds that the page manager has spent waiting to receive data from remote cache page servers.

PageReads (all)

The number of pages read by the process since it was last started. These page reads are actual disk reads and not reads from the shared page cache.

PageReadsProcessingCommit (Gem)

The number of pages read while the Gem is processing its part of the commit.

PageReadsStoneCommit (Gem)

The number of pages read while the Stone is processing its part of the commit.

PageReadsWaitingForCommit (Gem)

The number of pages read while waiting for the commit token.

PageServersInCacheCount (SPC monitor)

The total number of page servers attached to the shared cache.

PagesNeedReclaimSize (Stone)

The amount of reclamation work that is pending, that is, the backlog waiting for the GcGem reclaim task. Values greater than about 2000 are an approximation using increments of 65280.

PagesNotFoundInCacheCount (SPC monitor)

The number of pages needing to be removed from the cache that were not found in the cache. This statistic is updated by the cache page server on remote caches, or by the page manager on the Stone's shared page cache.

PagesNotRemovedFromCacheCount (Pgsvr)

The number of pages the page server was unable to remove from the cache. Requests to remove pages come from the stone. This statistic is updated by the cache page server on remote caches, or by the page manager on the Stone's shared page cache.

PagesRemovedDirtyFromCacheCount (SPC monitor)

The number of pages successfully removed from the cache by the cache page server or the Page Manager at the Stone's request. This statistic is updated by the cache page server on remote caches, or by the page manager on the Stone's shared page cache.

PagesRemovedFromCacheCount (SPC monitor)

The total number of pages successfully removed from the cache by the cache page server or the Page Manager at the stone's request.

PagesReturnedByGemsCount (Stone)

The total number of free pages returned to the Stone by any Gem.

PagesWaitingForRemovalInStoneCount (Stone)

The number of pages in the Stone that are waiting to be removed from the shared page cache by the Page Manager.

PageWaitQueueSize (Stone)

The size of the queue that holds sessions waiting to be allocated free pages. This queue should be empty or nearly so unless the repository is below its free space threshold.

PageWrites (all)

The number of pages written by the process since it was last started. These page writes are actual disk writes and not just writes into the shared page cache. Unless a large data load is in process, the number should be low for all processes except the Stone's AIO page server process.

PermGenSizeBytes (Gem)

The number of used bytes in the perm generation at the end of mark/sweep. Perm generation holds copies of Classes.

PersistentPagesCommittedCount (Gem)

The total number of pages made persistent (committed) by the session. This statistic is updated during commit processing.

PersistentPagesDisposed (Stone)

The number of persistent pages (pages already checkpointed) that have been disposed of while in the Stone's private cache.

PgsvrCheckpointState (Stone)

The state of checkpoint processing within an AIO pgsvr:

- 0=not active
- 1=writing dirty pages
- 2=finished write dirty
- 3=in fsync

PgsvrWaitQueueSize (Stone)

The number of remote sessions waiting on page server before reclaiming session resources.

PinnedDataPagesCount (SPC monitor)

The number of pinned data pages found in the cache.

PinnedOtPagesCount (SPC monitor)

The number of pinned object table pages found in the cache.

PinnedPagesCount (all)

The number of pages the process has pinned (locked) in the shared cache. Pages may be pinned by more than one process at the same time.

PinnedPrivatePagesCount (all)

The number of pages that the process has pinned (locked) in its private page cache.

PinnedSharedCount (SPC monitor)

The number of pages pinned by more than one process.

PinnedTotalCount (SPC monitor)

The total number of pinned pages found in the cache.

PomGenScavCount (Gem)

The number of times scavenge has thrown away the oldest pom generation space.

PomGenSizeBytes (Gem)

The number of used bytes in the pom generation at the end of mark/sweep. Pom generation holds clean copies of committed objects.

PossibleDeadKobjs (Stone)

The number of objects previously marked as dereferenced in the repository, but for which sessions currently in a transaction might have created a reference in their object space. (Units: 1000s of objects.) An object is not declared (“promoted to”) dead until each active session verifies the absence of such references during its next commit or abort.

PostCheckpointPages (Pgsvr)

The count of pages written out by the page server during post-checkpoint processing.

PreemptedBitmapPages (Pgsvr)

The number of bitmap pages removed from the shared page cache by this page server.

PreemptedCommitRecordPages (Pgsvr)

The number of commit record pages removed from the shared page cache by this page server.

PreemptedDataPages (Pgsvr)

The number of data pages removed from the shared page cache by this page server.

PreemptedObjectTablePages (Pgsvr)

The number of object table pages removed from the shared page cache by this page server.

PreemptedOtherPages (Pgsvr)

The number of pages removed from the shared cache by this page server that were not data, object table, commit record, or bitmap pages.

ProcessId (all)

The operating system processId for the process associated with this shared page cache process slot.

ProcessName (all)

This statistic identifies the process kind (Gem, Stone, page server, or shared page cache monitor).

ProgressCount (Gem)

You can use this statistic to monitor the progress of certain Repository methods that may run for extended periods.

This statistic is used in the following cases. For other cases, see the statistic **ProgressKobj**s on page 8-50.

- During `objectAudit`, `ProgressCount` is set to the number of live data pages at end of the object table scan, then decremented as data pages are scanned.
- During `_findPagesContainingOops`:, `ProgressCount` is set to the total number of pages in the repository and decremented as pages are processed.
- During `_readObjectTableFromOopNum:toOopNum:...`, `ProgressCount` is incremented as object table pages are read, reset to zero, then incremented again as `dataPages` (if any) are read.
- During `readPageRangeForGem:...`, `ProgressCount` is incremented as pages are read, then reset to zero.

ProgressKobjs (Gem)

You can use this statistic to monitor the progress of certain Repository methods that may run for extended periods. (Units: 1000s of objects.)

This statistic is used in the following cases. For other cases, see the statistic **ProgressCount** on page 8-49.

- During `markForCollection`, `ProgressKobj`s is incremented as live objects are marked during the marking phase, reset to zero, and incremented as possible dead objects are computed.
- During epoch garbage collection, `ProgressKobj`s is incremented as live objects within epoch are marked, then reset to zero.
- During `fullBackupTo:` and `restoreFromBackup:`, `ProgressKobj`s tracks the number of objects written to or restored from the backup file(s).
- During `objectAudit`, `ProgressKobj`s is incremented as the object table is scanned, then decremented as data pages are scanned.
- While reading and writing an FDC file of OOPs, `ProgressKobj`s is incremented as OOPs are read or written.

RcConflictCount (Gem)

The number of commits that resulted in an RC conflict.

RcRetryQueueSize (Stone)

The number of sessions waiting for the RcRetry object lock. (System waitForRcWriteLock:)

RebuildScavPagesForCommitCount (Gem)

The total number of times the gem rebuilt its list of scavengeable pages while processing a commit.

RecentActiveProcessCount (SPC monitor)

The number of active processes attached to the shared page cache. This statistic is computed more often than ActiveProcessCount and has decays quickly.

ReclaimCount (Stone)

The number of reclaims performed by a Reclaim GcGem process since the Stone repository monitor was last started.

ReclaimedPagesCount (Stone)

The number of pages reclaimed by a Reclaim GcGem process since the Stone repository monitor process was last started. The count indicates the number of pages that have been or will soon be placed back into the repository's pool of free pages.

RecoverTranlogBlockId (Stone)

The block ID of the transaction log currently being replayed during system recovery or restore.

RecoverTranlogFileId (Stone)

The file ID of the transaction log currently being replayed during system recovery or restore.

RemoteCachesNeedServiceCount (Stone)

The number of remote caches that require service from the Page Manager. Caches need service when they are starting or shutting down.

RemoteSharedPageCacheCount (Stone)

The total number of remote shared page caches attached to the system.

RepBkupRestPagesWrittenByGem (SPC monitor)
RepBkupRestPagesWrittenByStone (SPC monitor)

These statistics are obsolete; the page kind is no longer used.

ReposSizeMB (Stone)

The size of the stone's repository in MB.

Root40PagesWrittenByGem (SPC monitor)
Root40PagesWrittenByStone (SPC monitor)

The number of root 4.0 pages in the cache because of a write done by a Gem or by the Stone, respectively.

RootPagesWrittenByGem (SPC monitor)
RootPagesWrittenByStone (SPC monitor)

The number of root pages in the cache because of a write done by a Gem or by the Stone, respectively.

RunQueueSize (Stone)

The number of Gem session processes waiting for service from the Stone repository monitor.

ScavengeCount (Gem)

The number of scavenges executed by the in-memory garbage collector.

ScavengeOverflows (Gem)

The number of scavenges that overflowed in the in-memory garbage collector.

SessionId (all)

The GemStone sessionId associated with this client.

SessionStat0 — SessionStat47 (Gem)

These are computed by user code to define statistics associated with a session. There are 48 session cache statistic slots available. For details, see “Global Session Statistics” on page 8-22.

SessionWithGcLock (Stone)

The sessionId of the session holding the GcLock. A value of 1 indicates that the lock is held by Stone recovery or restore; 0 means that the lock is not issued.

ShadowedPagesCount (Gem)

The number of data pages added to the reclaim list due to commits by this Gem. This statistic is only updated during a commit.

SigAbortsReceived (Gem)

The number of times the Stone has signaled this session to abort, that it has received and recognized.

SigAbortsSent (Gem)

The number of times the Stone has signaled this session to abort, although the session may be in a sleep or I/O wait state and not yet aware of having received the signal. (See SigAbortsReceived (Gem), above.)

SleepDuringDisposeTempPageCount (Gem)

Total number of times the Gem slept while waiting to dispose of a temporary page. Each tick of the counter represents 5 milliseconds of sleep. If you encounter excessive counts, please report them to GemStone Technical Support.

SlotsCrashedCount (Stone)

The total number of slots for which the shared cache monitor has attempted recovery because a client process shutdown abnormally.

SlotsRecoveredCount (SPC monitor)

The total number of slots that the shared page cache monitor has recovered because a client process shutdown abnormally.

SpinLockCount (SPC monitor)

The current setting of the SHR_SPIN_LOCK_COUNT parameter.

SpinLockFreeFrameSleepCount (SPC monitor)

The number of times the process was forced to sleep on a semaphore while attempting to acquire the free frame list spin lock. Available only for shared page cache monitor slot.

SpinLockFreePceSleepCount (SPC monitor)

The number of times the process was forced to sleep on a semaphore while attempting to acquire the free page cache entry spin lock. Available only for shared page cache monitor slot.

SpinLockHashTableSleepCount (SPC monitor)

The number of times the process was forced to sleep on a semaphore while attempting to acquire a hash table spin lock. Available only for shared page cache monitor slot.

SpinLockNewSymSleepCount (SPC monitor)

The number of times a process was forced to sleep on a semaphore waiting for the NewSymbols spinLock.

SpinLockOtherSleepCount (SPC monitor)

The number of times the process was forced to sleep on a semaphore while attempting to acquire either the AllSymbols or shared counter spin lock. Available only for shared page cache monitor slot.

SpinLockPageFrameSleepCount (SPC monitor)

The number of times the process was forced to sleep on a semaphore while attempting to acquire a page frame spin lock. Available only for shared page cache monitor slot.

SpinLockSmcQSleepCount (SPC monitor)

The number of times the process was forced to sleep on a semaphore while attempting to acquire the SMC (shared memory communication) queue spin lock. The SMC queue allows Gems to communicate with the Stone process via shared memory. Available only for shared page cache monitor slot.

StnAioCompletionFailures (Stone)

The number of aio_write() operations for which either aio_error() or aio_return() reported that the AIO failed.

StnAioFsyncFailures (Stone)

The number of fsync() operations in AioWait thread which failed.

StnAioSuspendEAGAIN (Stone)

The number of times that aio_suspend() in AioWait thread returned EAGAIN error. A non-zero value indicates that some aio_write() operations are failing to complete within 15 seconds.

StnAioSuspendPrematureReturn (Stone)

The number of times that aio_suspend() returned prematurely with EINPROGRESS. This has been seen on AIX.

StnAioSyncWritesAfterCancel (Stone)

The number of AIOs initiated by aio_write() that failed to complete within 60 seconds, and were cancelled and written synchronously by AioWait thread. Should normally be zero.

StnAioWaitsForWork (Stone)

The number of times stone AioWait thread waited on semaphore for more work.

StnAioWriteEAGAIN (Stone)

The number of times aio_write() returned EAGAIN error due to lack of resources. Stat should normally be zero.

StnAioWriteFailures (Stone)

The number of aio_write() calls by stone Main thread which failed with other than EAGAIN. Stat should normally be zero.

StnGetLocksCount (Stone)

The total number of times the Stone retrieved the lock set and passed it to a remote gem.

StnLoopCount (Stone)

The total number of times the Stone has executed its service loop. If this number remains unchanged for a significant period (for example, ten seconds or so), the Stone has hung.

StnLoopHibernateCount (Stone)

The total number of times the Stone went to sleep waiting for a network event to occur. This state occurs when StnLoopState is set to 17.

StnLoopNoWorkThreshold (Stone)

Current setting of the STN_LOOP_NO_WORK_THRESHOLD stone configuration parameter.

StnLoopsNoWork (Stone)

Number of times the stone has executed its main service loop and found no work to perform. This counter is reset to zero each time the stone performs work.

StnLoopsSinceSleep (Stone)

Number of times the stone has executed its main service loop since sleeping. This counter is reset to zero each time the stone sleeps while waiting for work.

StnLoopState (Stone)

An integer that identifies where, in the Stone control loop, the Stone process is currently executing. For a meaningful statistic, set your sample rate to faster than a second. For state definitions, consult GemStone technical support.

StnMainWaitsForFreeAio (Stone)

The number of times the stone main thread waited for free AIO buffers to become available.

StnTranQToRunQThreshold (Stone)

Current setting of the STN_TRAN_Q_TO_RUN_Q_THRESHOLD stone configuration parameter.

StoneCommitState (Stone)

This internal statistic is used to determine the state of the Stone's commit processing.

SymbolCreationQueueSize (Stone)

The number of sessions waiting for a Symbol creation request to be processed.

TargetFreeFrameCount (SPC monitor)

The minimum number of unused page frames the free frame page server(s) will attempt to keep in the cache. The free frame count can still fall below this value if the cache contains mostly dirty pages, which free frame page servers cannot preempt.

TargetPercentDirty (Pgsvr)

The percent of dirty pages that AIO page servers try to maintain in the shared cache. If the dirty pages are below this target then the I/O rate will be limited on the next scan. If the dirty pages are above this target then the I/O rate will be set to AioRateMax.

TempObjSpacePercentUsed (Gem)

The approximate percentage of the total reserved temporary object memory for this session which is in use. Sessions are likely to encounter an out of memory error if this value approaches or exceeds 100%. This statistic is only updated at the end of a mark/sweep operation.

TempPagesDisposed (Stone)

The number of temporary pages (pages allocated since the last checkpoint) that have been disposed.

TimeInCommitRecordDisposal (Stone)

The total amount of real time in milliseconds that the Stone has spent disposing commit records. Commit records disposed during repository startup are not included in this statistic.

TimeInFramesFromFindFree (all)

The cumulative number of milliseconds that the Gem or Stone has spent scanning the shared page cache for a free frame since the session (for Gems) or Stone started.

TimeInMarkSweep (Gem)

The real time in milliseconds spent in in-memory garbage collector mark/sweeps.

TimeInPgsvrNetReads (Stone, Gem)

The cumulative amount of real time in milliseconds that a session or Stone has spent reading network data from a page server.

TimeInPgsvrNetWrites (Stone, Gem)

The cumulative amount of real time in milliseconds that a session or Stone has spent writing network data to a page server.

TimeInScavenges (Gem)

The real time in milliseconds spent in in-memory garbage collector scavenges.

TimeInStnGetLocks (Stone)

The total time spent by the Stone retrieving the lock set and passing it to a remote gem.

TimeInStonePageDisposal (Stone)

The elapsed real time in milliseconds the Stone spent performing page disposal operations.

TimeInUpdateUnionsCommit (Gem)

The total real time the gem spent updating its unions while waiting for the commit token.

TimeLastEpochGc (Stone)

Time at which the Admin GcGem finished computing dead objects for the last epoch. Time to end of the currently open epoch is computed from here.

TimePerformingCommit (Stone)

The number of milliseconds of real time the Stone has spent performing commit processing, not including time taken by asynchronous writes to the transaction log.

TimePerformingReadlo (Pgsvr)

The total amount of real time in milliseconds the page server has spent reading pages from disk.

TimePerformingReadRequests (Pgsvr)

The total amount of real time in milliseconds the page server has spent performing read requests for its client. This statistic includes time reading pages from disk and time searching the shared page cache for pages.

TimeProcessingCommit (Gem)

The cumulative amount of time in milliseconds that the Gem session process has spent doing the processing for commits while it has the commit token.

TimerThreadWakeups

The number of wakeups from sleep in stone Timer thread.

TimeStoneCommit (Gem)

The cumulative amount of time in milliseconds that the Gem session process has waited for the Stone repository monitor to complete commits by this session.

TimeWaitingForCommit (Gem)

The cumulative amount of time in milliseconds that the Gem session process has spent waiting for its turn to commit, that is, the time waiting for the commit token and the Stone's processing time for serialization.

TimeWaitingForIo (all)

The total real time in milliseconds that the process has spent waiting for I/O calls that read or write pages to complete. Only page I/O is included in this statistic. Other types of I/O (such as transaction log writes by the Stone process) are not included.

TimeWaitingForStone (Gem)

The total time the Gem spent waiting for a response from the Stone.

TimeWaitingForSymbols (Gem)

Cumulative elapsed time in milliseconds waiting for symbol creation requests to be processed.

TotalAborts (Stone)

The number of abort operations performed system-wide since the Stone was started.

TotalCommits (Stone)

The total number of commits (excluding read-only commits) performed by all processes since the Stone repository monitor was last started.

TotalGemFatalErrors (Stone)

The total number of fatal errors reported to the stone by all gems.

TotalLostOtsSent (Stone)

The total number of SigLostOtRoot signals sent by the stone.

TotalNewObjsCommitted (Stone)

The total number of new objects committed by all Gems.

TotalSessionsCount (Stone)

The total number of sessions currently logged in to the system.

TotalSessionsStopped (Stone)

The total number of stop session requests initiated by a gem or by the stone.

TotalSessionsTerminated (Stone)

The total number of sessions forcibly logged off by stone.

TotalSigAbortsSent (Stone)

The total number of SigAborts sent by the stone.

TrackedSetSize (Gem)

The number of objects in the Tracked Objects Set, as defined by the GCI.

TranlogFileId (Stone)

The file id of the transaction log to which the most recent tranlog entry was written.

TranlogRecordId (Stone)

The record id of the most recent transaction log entry to be written.

TranlogsFull

A flag indicating if all configured tranlog partitions or directories are full; 1 means full.

TransactionLevel (Gem)

This statistic describes the state of the Gem: 1 (in a transaction), 0 (outside a transaction), or -1 (a transactionless state in which the Stone will never signal the Gem to abort).

UpdateUnionsCommitCount (Gem)

The total number of times the gem updated its unions while waiting for the commit token. This count will be at least one for every commit.

VoteNotDead (Gem)

The number of objects that the Gem process removed from the possible dead set the last time that it voted on the possible dead.

VoteNotDeadKobjjs (Stone)

The number of objects that the Gem process removed from the possibleDead set the last time that it voted on the possible dead. (Units: 1000s of objects.)

WaitingForSessionToVote (Stone)

The sessionId of a session which the system is waiting for to complete the voting on possible dead objects. A zero value indicates that it is not waiting.

WaitsForOtherReader

The number of PageRead operations avoided by waiting for read already in progress by another process.

WorkingSetSize (Gem)

The number of objects in memory that have an object Id assigned to them. Approximately the number of committed objects that have been faulted in plus the number that have been created.

—
|

Making and Restoring Backups

This chapter explains describes how to make backups of your GemStone/S 64 Bit repository and, should it become necessary, how to use the backups to restore the repository.

This chapter includes the following topics:

- How to Make an Online Extent Backup (page 9-3)
- How to Restore from an Online Extent Backup (page 9-3)

An online extent backup is, essentially, a snapshot copy of the repository extents with the system running. Online extent backup is the primary means of performing repository backups and can be run during production hours.

- How to Make a Smalltalk Full Backup (page 9-8)
- How to Restore from a Smalltalk Full Backup (page 9-14)

As an alternative, you can perform full backups during non-production hours, using backup and restore methods provided as part of the GemStone kernel. Full backups are required if you want to reduce the number of extents in the repository.

- How to Make an Offline Extent Backup (page 9-33)
- How to Restore from an Offline Extent Backup (page 9-33)

It is also possible to use an offline extent file, using a snapshot obtained while GemStone is shut down.

- How to Recover After Repair of the File System (page 9-3)

This section presents procedures for recovery from a disk failure or corrupted file system.

NOTE

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 1-11.

- How To Manage a Warm Standby System (page 9-41)

To maximize system availability, you can run a second GemStone installation on standby, ready to take over in case of any failure of the primary system.

9.1 Overview

One way of safeguarding your repository is to create a GemStone backup periodically and then store the backup in a secure place. It's best to establish a regular backup schedule that fits your application and to keep system users informed of that schedule.

Also back up transaction logs, especially if you have enabled full transaction logging. These logs allow you to roll forward from a backup to the state of the last committed transaction. Transaction logs in the file system can be backed up as part of a regularly scheduled system backup using operating system utilities. For a related discussion, see “To Archive Logs” on page 7-8.

Because backup files do not include the object table, they are typically 15–20% smaller than the running repository, not including any free space in the extents.

9.2 How to Make an Online Extent Backup

Privileges required: SystemControl.

An online extent backup is, essentially, a snapshot copy of the repository extents with the system running. Online extent backup is the primary means of performing repository backups and can be run during production hours.

Three steps are involved in an online extent backup:

1. Suspend checkpoints.

Checkpoints are not permitted while the online backup is in progress. There must not be a checkpoint in progress when the online backup begins, and no checkpoints are allowed to begin until the online backup has finished. All other database operations (including commits and aborts) are permitted during the online backup.

To suspend checkpoints for a specified number of minutes, call `System class >> suspendCheckpointsForMinutes::`. If this method is called while a checkpoint is in progress, it will block until the current checkpoint completes. 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.

You cannot suspend checkpoints while in partial transaction log mode or while the repository is in restore mode. However, you can start a new transaction log while checkpoints are suspended. To do so, call `Repository >> startNewLog`.

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 as well as an Integer that indicates the number of seconds remaining in the suspension.

For example:

```
topaz 1> printit
System checkpointStatus
%
an Array
  #1 false
  #2 0
```

```
topaz 1> printit
System suspendCheckpointsForMinutes: 15
%
true
```

```
topaz 1> printit
System checkpointStatus
%
true 900
```

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.

3. Resume checkpoints.

Once the extent copy has completed, a session will 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
%
false
```

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.

If the system is shut down while checkpoints are suspended, checkpoints will be reenabled and a final checkpoint will be written during the clean shutdown process. Any online backups in progress during system shutdown must be discarded.

Methods to Perform Checkpoints

To support the use of checkpoints, the following methods in class System (category Online Backup Support) are provided:

`suspendCheckpointsForMinutes`: *minutes*

Suspends all checkpoints for the given number of minutes or until the `resumeCheckpoints` method is executed, whichever occurs first. Calling this method while checkpoints are already suspended changes the duration of the suspension. If a checkpoint is in progress when this method is called, the call blocks until the current checkpoint completes, at which time checkpoints are suspended. If any session has made this call and is waiting for the current checkpoint to complete, calls to this method by other sessions will fail. Returns true if checkpoints were successfully suspended.

`resumeCheckpoints`

Resumes regular checkpoints if they were previously suspended by `suspendCheckpointsForMinutes`. Returns true if checkpoints were previously suspended, returns false if checkpoints were not suspended.

`checkpointStatus`

Returns an Array of two elements: a Boolean that indicates if checkpoints are currently suspended, and an Integer indicating the number of seconds before checkpoints will be resumed by the stone.

In addition, two methods in class System (category Transaction Control) are provided for starting checkpoints:

`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. Does 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. Does not commit, abort, or otherwise modify the current transaction.

An Example Script

As part of a complete backup strategy, you can create an online extent backup script for your system. Your GemStone installation directories include the example script `$GEMSTONE/examples/admin/onlinebackup.sh`. You can customize this script for your own system; you must add the necessary code to perform the file system copies of your extents.

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.

As shown in `onlinebackup.sh`, your script must perform the following actions:

1. Suspend checkpoints. For example:

```
topaz 1> printit
System suspendCheckpointsForMinutes: minutes
%
```

2. Perform a file system backup of live extents.

The actual backup may be as simple as using the UNIX `cp` command for each of the extent files. However, on production systems, this may involve breaking a disk mirror and subsequently resynchronizing. The example script does not include code for this step.

3. Resume checkpoints after the backup.

```
topaz 1> printit
System resumeCheckpoints
%
```


9.3 How to Restore from an Online Extent Backup

To restore the repository from an online extent backup to the last committed transaction, the online backup files must be available, along with all transaction logs written since the backup was started. (The previous transaction log may also be required.)

The restore procedure includes these steps:

1. Copy the extents from the backup to the location where the repository extents reside. This is essentially the reverse of Step 2 on page 9-4.
2. Use **startstone -R -N** to restart GemStone. These options start the Stone in restore mode, but do not attempt to automatically recover by replaying all transaction logs. (For more about these options, see page B-15.)
3. Restore all transaction logs written since the online extent backup was performed. You can query the Stone to determine the sequence number of the first transaction log required for the restore.

Each restore operation, on completion, terminates its GemStone session. You will need to log in again before performing the next restore operation.

```
topaz 1> printit
SystemRepository restoreStatusOldestFileId
%
6
```

```
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
Restore from transaction log(s) succeeded.
```

4. When all transaction logs have been restored, commit the restore.

```
topaz 1> printit
SystemRepository commitRestore
%
Restore from transaction log(s) succeeded. commitRestore
succeeded
```

5. The repository is now ready for use.

9.4 How to Make a Smalltalk Full Backup

Privileges required: FileControl.

As an alternative to online extent backups, you can perform Smalltalk full backups during non-production hours, using backup and restore 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 copied and will be restored, improving performance of such routine maintenance tasks as garbage collection.

In a Smalltalk full backup, the method `Repository>>fullBackupTo:` saves the most recently committed version of the repository in a way that is consistent from a transaction viewpoint. The method `fullBackupTo:` forces 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. (An alternate method, `Repository>>fullBackupTo:mBytes:`, lets you create a multiple-file backup by limiting the size of each part.)

As with online extent backups, 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.

With full transaction logging enabled, all work committed since the backup is saved in a set of transaction logs. In the event of a subsequent repository failure, the backup file together with the set of logs created since the backup contain all information necessary to restore the repository to its current committed state. In the absence of full transaction logging, a media failure can cause the loss of all updates since the last backup.

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. However, this step usually lasts ten minutes or less.
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 10-2.)

3. In the final step, all 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. For most systems, this step takes the longest of the three.

The fullBackupTo: methods

```
Repository>>fullBackupTo: fileOrDevice
```

```
Repository>>fullBackupTo: fileOrDevice MBytes: mByteLimit
```

fileOrDevice specifies the file or device where the backup is to be created. You must specify the name of a file or device, not a directory name.

You can create backups on a remote node by using a network resource string (NRS) to specify the node name as part of *fileOrDevice*. For an example, see “To Create a Backup on a Remote Node” on page 9-11.

WARNING

If the specified fileOrDevice runs out of space, such as off the end of a tape, the backup will terminate with a system I/O error at that point, and will be unusable. To avoid having to repeat the entire backup, make sure that the device has sufficient space or set mByteLimit appropriately.

mByteLimit lets you create a multiple-file backup by limiting the size of each part. This argument is especially useful when the size of the backup exceeds the capacity of a single tape. For further information and an example, see “To Create a Backup in Multiple Files” on page 9-12. If you don’t want to limit the size of the backup file, specify a *mByteLimit* of 0 or use the simpler

Repository>>fullBackupTo: message, omitting MBytes: entirely. A value of *mByteLimit* less than 0 or greater than 4096000 generates an error.

NOTE

We recommend that individual backup files be no greater than 16 GB. If your repository backup would be greater than 16 GB, use the MBytes: argument to create four or more backup files.

To perform a Smalltalk full backup, send your repository the message `fullBackupTo: fileOrDevice`. For example:

```
topaz 1> printit
"Create a full backup of SystemRepository in the file
  '/users/backups/march12.07'"
SystemRepository fullBackupTo: '/users/backups/march12.07'
%
true
```

The backup file named `march12.07` is created.

During the backup, 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, send `System beginTransaction` or `System transactionMode: #autoBegin`.

If the backup file already exists, or if the *fileOrDevice* argument is an empty string, the method returns an error.

Additional Performance Tips

For the session performing the backup, the statistic `ProgressCount` (described on page 8-49) indicates the number of objects written to the backup file thus far. If you know the number of objects in the repository, you can use this statistic to determine how far the backup has progressed.

Backup and restore operations accessing tape drives are usually slower than those accessing a hard disk.

You can often improve both backup and restore performance by increasing the size of the shared page cache.

You can usually improve restore performance if you also increase the size of the private page cache for the Gem performing the restore. For example, in that Gem's configuration file, include the line:

```
GEM_PRIVATE_PAGE_CACHE_KB = 65536;
```

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 10-2.

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` (page 8-46). If that statistic is high, run one or more `Reclaim GcGems` before performing the backup. (See “Admin and Reclaim GcGems” on page 10-8.)

Dead objects waiting to be reclaimed (measured by the statistic `DeadNotReclaimedKobjs`, described on page 8-29) are not backed up, as these objects are going to be deleted anyway.

Possibly dead objects are included in the backup file. (Possibly dead objects are defined in “What Happens to Garbage?” on page 10-6). However, the possible dead set is not backed up. So if a `markForCollection` or other garbage-marking operation completed before the backup, but the possibly dead objects had not yet been promoted to dead, the garbage-marking operation will have to be repeated.

To avoid this, therefore, if you back up your repository after a `markForCollection` or other garbage-marking operation, wait until the statistic `PossibleDeadKobjs` (page 8-48) falls, and the statistic `DeadNotReclaimedKobjs` (page 8-29) rises.

To Create a Backup on a Remote Node

The following example uses an NRS to access a tape drive located on another node. You can use the same approach to access a remote disk. Of course, performing a backup across the network is likely to take much longer than writing it to a local device.

```
topaz 1> printit
"Access a tape device on node flute"
SystemRepository fullBackupTo:'!@flute!/dev/rst0'
%
```

A GemStone NetLDI must be running on the remote node. The user performing the backup must provide authentication for that node, such as an entry in a `.netrc` file. The requirements are similar to those given in Chapter 3 for starting an RPC Gem session process on a remote node.

To Create a Backup in Multiple Files

To create a multiple-file backup, include the argument `MBytes: mByteLimit`. For example, `MBytes: 10` tells GemStone to limit the size of the backup file to 10 MB. When backing up to tape, use the tape's available space (in MB) as the `MBytes:` argument.

NOTE

Writing a backup to multiple files takes longer than writing it to one file.

If a backup requires more space than you specified in `mByteLimit`, the backup stops after creating one file and returns a result similar to this:

```
topaz 1> printit
"Start a full backup of SystemRepository in the file
  '/users/backups/march12.07'"
SystemRepository fullBackupTo: '/users/backups/march12.07'
MBytes: 10
%
Finished writing backup file 0 of a multifile backup
```

To create the next file in the backup, send your repository the message `continueFullBackupTo: fileOrDevice MBytes: mByteLimit`.

For example:

```
topaz 1> printit
"Continue a full backup by writing a second file to
  '/users/backups/march12.07_2'"
SystemRepository continueFullBackupTo:
'/users/backups/march12.07_2' MBytes: 10
true
```

If the backup is suspended because it reached the specified byte limit, the session may or may not be in a transaction, depending on how far the backup has progressed. The `continueFullBackupTo:` method operates properly in either case.

Commits and aborts by the session doing a multiple-file backup are disallowed until the backup completes. If you need to cancel the backup, use the method `Repository>>abortFullBackup`. The session can then commit or abort its current transaction.

When backing up to multiple disk files, be sure to specify a unique file name for each continuation. If you need to verify the file sequence later, each file contains a sequential `fileId`, which you can examine using `copydbf -i fileName`.

To Create Compressed Backups

It is possible to write and read full backup files in compressed mode.

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` if you do not specify that suffix and if the backup is being written to a file system file.

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 with `gzip`, is readable by

```
restoreFromBackup:.
```

NOTE

*Transaction logs are always written in uncompressed format. These files may be compressed with **gzip** before archiving them to tape or to other disks. Such archived tranlogs can be restored directly, without having to run **gunzip** on them, although the process is less efficient.*

To support compression, the following methods in class `Repository` (category `Backup and Restore`) are provided:

```
continueFullBackupCompressedTo: fileOrDevice MBytes: mByteLimit
```

This method is similar to `continueFullBackupTo:MBytes:` except that the output file is written compressed in `gzip` format. The output file must be on a local file system or accessible via NFS. Backup files written to a file system in compressed mode are automatically appended with the suffix `.gz` if that suffix is not specified by the user.

```
fullBackupCompressedTo: fileOrDevice
```

This method backs up the receiver to a single backup file or tape in `gzip` format. The output file is written compressed in `gzip` format and cannot be written to a raw device. The output file must be on a local file system or accessible via NFS. Backup files written to a file system in compressed mode are automatically appended with the suffix `.gz` if that suffix is not specified by the user.

```
fullBackupCompressedTo: fileOrDevice MBytes: mByteLimit
```

This method is similar to `fullBackupTo:MBytes:` except that the output file is written compressed in `gzip` format. See `fullBackupCompressedTo:`

To Verify 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/march12.07 /dev/null
```

To Examine the Backup Log

The path of the backup file and starting time are written to `UserGlobals at:#BackupLog`. To see a listing of previous backups performed by the current `userId`, execute the following (which returns an error if the repository has never been backed up):

```
topaz 1> level 2
topaz 1> printit
BackupLog
%
fullBackup to /users/backups/march12.07 started at
Mar/12/2007 13:22
```

9.5 How to Restore from a Smalltalk Full Backup

Privileges required: FileControl.

NOTE

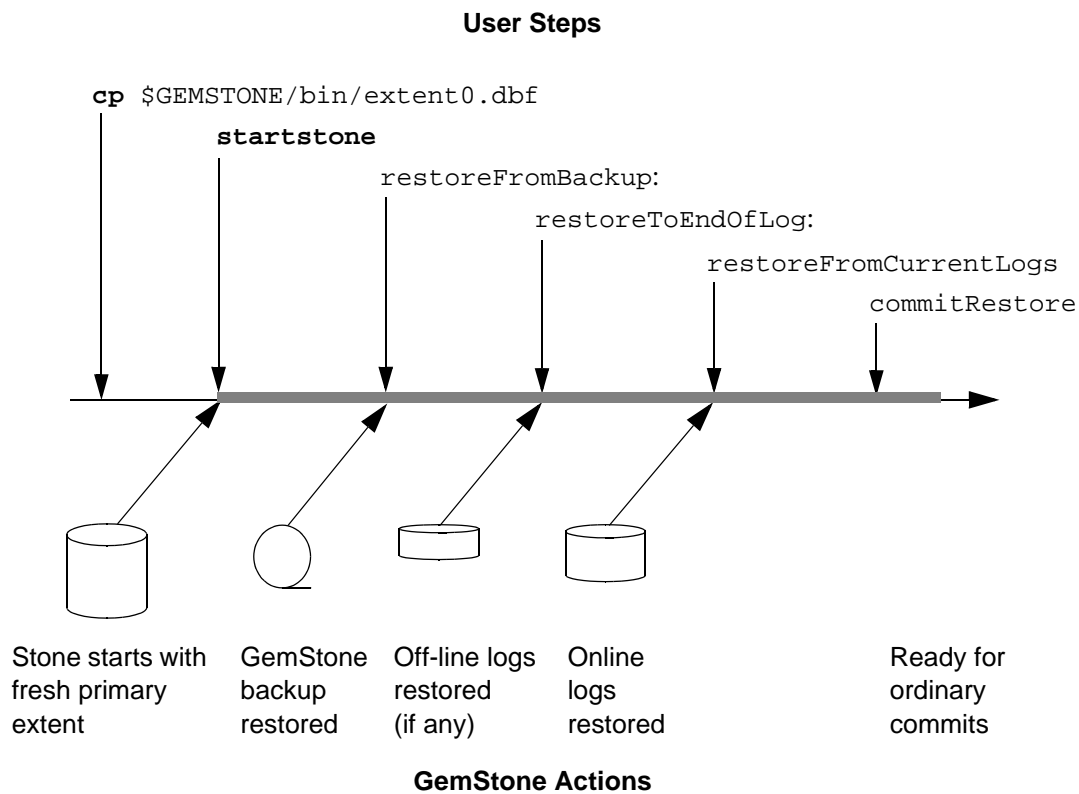
To avoid failure of the restore operation, after a repository conversion or upgrade, you must run the restore from the SystemUser account.

Restoring the repository from a Smalltalk full backup ordinarily takes place in two phases:

1. Restore the repository from the last GemStone backup file or tape.
 - For a step-by-step description, see “Phase 1: Restore to the Point of the Backup” on page 9-16.
2. Apply transaction logs to restore transactions that were committed after the backup was started. (The backup must have been made while the repository was in full transaction logging mode.)
 - For a step-by-step description, see “Phase 2: Restore Subsequent Transactions” on page 9-19.

Figure 9.1 illustrates the restore process for Smalltalk full backups.

Figure 9.1 System Timeline: Restoring from a Smalltalk Full Backup



Before you begin, make sure you have a Smalltalk full backup of the system repository, created by GemStone with the method `fullBackupTo:.` (See “How to Make a Smalltalk Full Backup” on page 9-8.)

If full backups of your repository require more than one tape, restoring tape 1 without tape 2 does not give you a usable version of the repository. The objects on tape 1 are copied, but the transaction cannot be committed until you restore tape 2. (The Topaz **commit** command does not commit a partial restore, even though the message may say the commit was successful.)

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,

but not identical, to their organization in the original repository. This clustering reflects both explicit clustering of objects by the application and default clustering into the generic “don’t care” cluster bucket.

If the number of extents during restoration is the same as when the backup was started, such clustering in the original repository takes precedence over the `DBF_ALLOCATION_MODE` configuration setting used by the Stone performing the restore operation. 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.

Phase 1: Restore to the Point of the 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.

NOTE

Make sure that you have full backups of good repository files. 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, perform the following procedure:

Step A1. If GemStone is still running, tell all users to log out and use **stopstone** to stop the system. Certain file system failures while the Stone is running may make it necessary to use **kill processid** to kill the Stone process.

Step A2. 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.

WARNING

Do NOT delete the transaction log files up to the time of the crash—leave them online in their current locations.

Step A3. Delete all extent files specified in `DBF_EXTENT_NAMES` in your configuration file.

Step A4. Copy the distribution 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 location of the *new (second)* extent; the Stone repository monitor will create the new extent at startup.

We recommend that you use the GemStone **copydbf** command to create the copy, rather than using the UNIX **cp** command; **copydbf** must be used if you are copying to or from a raw partition.

For example:

```
% copydbf $GEMSTONE/bin/extent0.dbf \  
$GEMSTONE/data/extent0.dbf  
% chmod 600 $GEMSTONE/data/extent0.dbf
```

Use **chmod** to give the copy the same permissions you ordinarily assign to your repository files.

Step A5. Ensure that there is space to create a log file during recovery. 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.

GemStone starts a new transaction log file when you start the restore process (Step A8). In addition, you can force closure of the current log and opening of

a new log at any time by using the method `Repository>>startNewLog`. That method returns a `SmallInteger`, which is the *fileId* of the new log.

Step A6. Use `startstone` to restart the Stone.

Step A7. 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 A8. Restore the most recent full backup to the new repository by sending the message `restoreFromBackup: fileOrDevice`. The argument *fileOrDevice* can be the name of a file or tape device. This method automatically detects whether a backup is compressed or not and reads it accordingly.

The following example restores the repository from a backup that consists of a single disk file.

- For information about using tapes, see “To Restore Backups from Tape” on page 9-24.
- For information about restoring backups from more than one file, see “To Restore Multiple-File Backups” on page 9-23.

The message `restoreStatus` can return helpful information at any point in the restore process. This status is an attribute of the repository, not of the session, and persists across login sessions and stopping and starting of the Stone repository monitor.

```
topaz 1> printit
SystemRepository restoreStatus
%
Restore is not active
topaz 1> printit
SystemRepository restoreFromBackup: 'backup.dat'
%
```

If the full backup is contained in one file, the system commits the restore and returns a summary and status. For instance:

```
[restore info]:GemStone attempting to open backup.dat .  
[restore info]:GemStone reading from      backup.dat .  
The restore from full backup completed, with  
SmallInteger(40417) objects restored and  
SmallInteger(0) corrupt objects not restored.
```

- ❑ 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 full logging was in effect (`STN_TRAN_FULL_LOGGING = true`), the status line is similar to this:

```
Ready for restore from transaction log(s).
```

Continue with “Phase 2: Restore Subsequent Transactions” on page 9-19.

CAUTION

Although you can end the restore process before restoring from all transaction logs, doing so can make it impossible to restore the omitted logs later by repeating the process. If you plan to terminate the restore prematurely, first read “Precautions When Restoring a Subset of Transaction Logs” on page 9-30.

Phase 2: Restore Subsequent Transactions

If full transaction logging was in effect, the second phase of restoring the repository is to roll forward from the state 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 can only restore transactions committed within a single backup-restore cycle; that is, the transactions being restored cannot span a more recent restore.

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, first

read “Precautions When Restoring a Subset of Transaction Logs” on page 9-30.

Step B1. Before continuing the restore process, you must log in again. (The `restoreFromBackup`: method (Step A8) terminated the session when it completed.)

```
topaz> login
```

Step B2. Determine the location of all needed transaction logs. The method `restoreStatus` identifies the oldest transaction log that is needed, or you can use `copydbf -i backupName`. If a session was in a lengthy transaction at the time of the backup, that log may be one that was written before the backup started. In this example, it is `tranlog6.dbf`:

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 03/02/07 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`.

The methods in category Configuration File Access of class System allow you to inspect the log directory paths and file name prefixes for the current configuration. (For information, see “How to Access the Server Configuration at Run Time” on page 1-38.)

Step B3. Restore the transaction logs in time sequence, beginning with the log identified by `restoreStatus`. How you restore the logs depends on where the oldest logs are located. If you encounter a failure because of a truncated or corrupted transaction log, refer to “Errors While Restoring Transaction Logs” on page 9-26.

- If all transaction log files beginning with `tranlogfileId.dbf` are in the locations specified by the `STN_TRAN_LOG_DIRECTORIES` configuration option, skip to Step B4.
- If several of the older transaction logs have been moved to a different disk location, send the messages

Repository>>setArchiveLogDirectories: and
 Repository>>restoreFromArchiveLogs to restore those log files.
 The following example restores logs archived in /GS-archive, which is
 not one of the active locations listed in STN_TRAN_LOG_DIRECTORIES:

```
topaz 1> printit
SystemRepository setArchiveLogDirectories:
  #( 'GS-archive' )
SystemRepository restoreFromArchiveLogs
%
```

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.

Continue with the next substep if it applies; otherwise continue with Step B4.

- If individual transaction logs need to be restored, execute
 Repository>>restoreToEndOfLog: *fileId* for each file or raw partition
 in time sequence. (If the files are on tape, the files must first be restored to
 a disk drive, but they do not need to be in their original location.) For
 example:

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 03/02/07 13:51:19 PST
  next fileId = 7, record = 10.
topaz 1> printit
SystemRepository restoreToEndOfLog: '%/tranlog7.dbf'
%
Restore from transaction log succeeded.
```

As mentioned previously, each restore operation, on completion,
 terminates its GemStone session. You will need to log in again before
 performing the next restore operation.

Continue with Step B4 to restore from online logs.

- Step B4.** Restore transactions from all remaining online log files by executing the
 method Repository>>restoreFromCurrentLogs. The remaining log
 files (all log files beginning with the *fileId* currently returned by

restoreStatus) must be online and in the directories or raw partitions specified by STN_TRAN_LOG_DIRECTORIES.

```
topaz> login
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
Restore from transaction log(s) succeeded. [null]
```

Step B5. Send the message `commitRestore`, which tells the system you are finished restoring transaction logs.

```
topaz> login
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository commitRestore
%
Restore from transaction log(s) succeeded. commitRestore
succeeded
```

At this point, restore mode is no longer active, and no further logs can be restored. Logins have been enabled, and ordinary user commits will be allowed.

If you send `commitRestore` earlier in the restore process (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.

This step completes the restore process. Make a new full backup as soon as operational circumstances permit.

Other Considerations

Performance Tips

Restoring from backup usually takes at least 10-30% longer than the full backup took.

For the session performing the restore, the statistic `ProgressCount` (described on page 8-49) indicates the number of objects restored from the backup file thus far. If

you know the number of objects in the backup file, you can use this statistic to determine how far the restore has progressed.

You can improve restore performance by using one page server for every extent that resides on its own dedicated disk drive, but only if the repository uses weighted extent allocation mode. For example, if you are restoring a repository with six extents, each on its own separate physical hard drive, the following parameters, set in the restoring Gem's configuration file for the duration of the restore, would improve performance:

```
STN_NUM_LOCAL_AIO_SERVERS = 6;  
DBF_ALLOCATION_MODE = 10,10,10,10,10,10;
```

Here are some tips on getting restores completed as soon as possible:

- Use `restoreFromBackups:` or `restoreFromBackupsNoShadows:` methods, rather than restoring backups one by one. These methods require you to specify all backup files in the correct order in an Array.
- A large shared page cache will improve performance.
- Set the Gem's free frame limit to be very low (for example, 500).
- Restore from uncompressed files rather than compressed files. The decompression libraries are slow and do very inefficient I/O. Backups created from `fullBackupCompressedTo:` can be uncompressed using **gunzip** *before* the restore operation for best performance. This makes a big difference.
- Place extents on striped file systems.
- Make sure the backup files being restored are not on the same disk spindles as the extents or transaction logs.

To Restore Multiple-File Backups

If you are restoring a backup that occupies more than one file or tape, each backup file must be restored in sequence. You can determine the restore status and the internal identification of the backup file required next by using the method `restoreStatus`. If a restore process is active, the reply indicates the date and time to which the repository has been restored, and the type of file and the internal file identification number (*fileId*) of the file that must be restored next. (The *fileId* is reset to 0 at the beginning of each Smalltalk full backup.)

For each part of the backup, repeat Step A8 on page 9-18, using the next file in sequence. If you are uncertain of the sequence in which to restore a particular file, use `copydbf -i fileName` to display its *fileId*.

Another way is to use `restoreFromBackups: arrayOfFilesOrDevices` and include all files or devices in the array in proper sequence. Since all backup files are verified for integrity at the beginning of the restore, they all must be available when this method is called. The following example restores a backup that occupies two files by placing the filenames in an array:

```
topaz 1> printit
SystemRepository restoreFromBackups:
#( '/backups/Jan_1.1'
  '/backups/Jan_1.2' )
%
Opening file /backups/Jan_1.1
Opening file /backups/Jan_1.2
Restore from full backup completed with 132804 objects
restored and 0 corrupt objects not restored.
```

Another alternative is to use `restoreFromBackupsNoShadows: arrayOfFilesOrDevices`. This method is the same as `restoreFromBackups:` except that partially filled data pages are not added to the list of scavengeable pages upon completion of the restore. Using this method may cause the restored repository to perform faster than one restored using `restoreFromBackups:`, especially immediately following the restore.

When you restore the last backup file, GemStone either commits the restore or indicates that it is ready to restore from transaction logs, as explained on page 9-19.

If an error occurs, the shadow object space being built by the restore reverts to its state as of the end of the last file that was successfully restored.

If you need to cancel the process and start over while restoring a multiple file backup, use the method `Repository>>abortRestore`.

If the Topaz login session dies before the last backup file has been restored, you must start over by restoring the first file of the set.

To Restore Backups from Tape

When restoring from tape, use the device name as the argument to `SystemRepository restoreFromBackup:.` For example:

```
topaz 1> printit
SystemRepository restoreFromBackup: '/dev/rmt0.4'
%
```

Depending on your operating system and the type of tape drive, you may need to issue operating system commands first. For information, check your operating system documentation.

To access a tape device on another host, make sure that a NetLDI is running on that host and then include the host's name in *fileOrDevice* as a network resource string. For instance:

```
topaz 1> printit
SystemRepository restoreFromBackup: '!'@flute!/dev/rst0'
%
```

To Restore Logs to a Point in Time

Ordinarily, the methods `Repository>>restoreToEndOfLog:` and `restoreFromCurrentLogs` restore all transactions in the log file. However, you can specify an earlier stopping point by sending the message `timeToRestoreTo: aDateTime`. Restoration will stop at the first repository checkpoint that originally occurred at or after *aDateTime*.

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 maximum 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)
  The file was created at:      03/25/07 14:55:59 PDT.
  The previous file last recordId is 69.
Destination file: /dev/null
  byteOrder: Sparc (MSB first)
  Checkpoint 1 (last in file) started at: 03/25/07 14:55:59
PDT.
  oldest transaction references fileId -1 ( this file ).
  File size is 12288 bytes (24 records).
```

The following sequence restores the repository to the first checkpoint that would have included a commit on March 22, 2007 at 3:35:00 p.m.:

```
topaz 1> printit
SystemRepository timeToRestoreTo:
  (DateTime fromString: '22/03/2007 15:35:00') .
SystemRepository restoreFromCurrentLogs
%
```

You can continue restoring past *aDateTime* by issuing another `restoreToEndOfLog:` or `restoreFromCurrentLogs`. If you first issue another `timeToRestoreTo:`, restoration stops at the new *aDateTime*; otherwise, restoration continues to the end of the log.

Errors While Restoring Transaction Logs

Sometimes a transaction log is inadvertently truncated or corrupted. This section discusses types of errors that can occur.

Disk-full error during a copy operation

An unnoticed disk-full error during a copy operation can result in a truncated log that goes undetected until you try to restore from it. Because of the way transaction logs are written, a truncated log may appear to restore properly, and the gap will not be detected until the next log is read.

If a message like the following appears:

```
Restore from transaction log failed.
Reason: Log with fileId 37 is truncated or corrupt.
Continue restore with complete copy of this log.
```

check the *fileId* in the message to identify the log that caused the problem, which may be either the log currently being restored or the previous one. If the transaction log is a file (not a raw partition), its default name is `tranlogfileId.dbf`.

The method `restoreStatus` returns additional information indicating the next record expected within the current log file. For instance:

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to 03/02/07 13:26:31 PST
next fileId = 6, record = 9.
```

Retrieve a copy of the transaction log from a UNIX backup, and complete restoring it by using `restoreToEndOfLog : fileId`. Then continue the restore process by using either `restoreFromCurrentLogs` or another `restoreToEndOfLog :`.

Missing transaction log file

If a transaction log file in the sequence is missing, `restoreFromCurrentLogs` stops at that point. For example, if `tranlog4.dbf` is missing, `restoreFromCurrentLogs` stops after restoring from `tranlog3.dbf`.

You can execute the method `restoreStatus` to identify the next log file expected. Ensure that the missing file(s) are present and then continue the restore process with `restoreFromCurrentLogs` or `restoreToEndOfLog :`.

In the following example, `restoreFromCurrentLogs` encounters a truncated log. Invoking `restoreStatus` confirms that `tranlog6.dbf` is incomplete and shows that restoration needs to continue with record 9:

```
topaz 1> printit
SystemRepository restoreFromBackup: 'backup.dat'
%
[restore info]:GemStone attempting to open backup.dat .
[restore info]:GemStone reading from      backup.dat .
The restore from full backup completed, with
SmallInteger(40483) objects restored and
  SmallInteger(0) corrupt objects not restored.
Ready for restore from transaction log(s).
topaz> login
topazm: in login, found topaz session 1 stale
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
-----
GemStone: Error      Fatal
Restore from transaction log failed,
  Log with fileId 6 is truncated or corrupt, or log 7 is
corrupt.
Repeat restore of complete copy of log 6, then restore log
7.
Error Category: [GemStone] Number: 4049 Arg Count: 1
Arg 1: 20
topaz> login
topazm: in login, found topaz session 1 stale
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 03/02/07 13:26:31 PST
  next fileId = 6, record = 9.
```

To recover, restore a complete copy of that transaction log by name (here, "tranlog6.dbf"):

```
topaz 1> printit
SystemRepository restoreToEndOfLog: 'tranlog6.dbf'
%
Restore from transaction log(s) succeeded. [null]
```

```
topaz> login
topazm: in login, found topaz session 1 stale
gci login: currSession 1 rpc gem processId -1
successful login
```

```
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 03/02/07 13:26:42 PST
  next fileId = 7.
```

The restore status now shows that you are ready for the next transaction log, tranlog7.dbf. Because the remaining logs are online, return to the original procedure of restoring them as a group and then commit the restored repository:

```
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
Restore from transaction log(s) succeeded. [null]
```

```
topaz> login
topazm: in login, found topaz session 1 stale
gci login: currSession 1 rpc gem processId -1
successful login
```

```
topaz 1> printit
SystemRepository commitRestore
%
Restore from transaction log(s) succeeded. commitRestore
succeeded
```

If you cannot find an undamaged copy of the transaction log, `commitRestore` will commit as much as has been restored. However, if there is any chance of a finding a good copy, see the following discussion, "Precautions When Restoring a Subset of Transaction Logs".

Precautions When Restoring a Subset of Transaction Logs

If, for some reason, you want to end the restore process before restoring transactions from one or more log files, be aware of the likely consequences and some precautions that may be appropriate:

- Obviously, the omitted transactions will be lost. Unless the omitted log is missing, presumably that is your intent.
- Less obvious, where the omitted logs actually are present, is that it may be difficult or impossible to reverse your action later and restore the omitted logs by repeating the entire restore process. Operations after the first restore create a time fork in the repository, and attempting to reverse the course later results in object audit errors. (For an example of this, see the following discussion, “Fork-in-Time” Scenario.)

If there is any chance that you may want to restore from the omitted transaction logs later, modify the ordinary restore procedure in this way:

1. Before starting the Stone on the fresh extent (Step A8, on page 9-18), move all transaction logs to another directory.
2. After restoring from the backup (`restoreFromBackup:`), restore transaction logs individually by using `restoreToEndOfLog:` (Step B2, on page 9-20) and providing the new path.

Repeat `restoreToEndOfLog:` for each log file you want to restore.

3. Send the message `commitRestore` to end the restore operation.

CAUTION

If you subsequently decide to restore logs by repeating the entire process, first remove any new log files since the start of the restore process, and then move the previous transaction logs back to their original location. Follow the ordinary restore procedure (page 9-14). Note, however, that you will not be able to restore any transactions committed after your initial `commitRestore`. That work will be lost.

“Fork-in-Time” Scenario

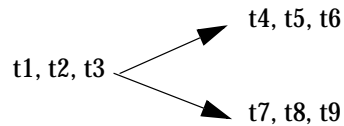
In some cases, you may encounter problems when there you cannot restore from your most recent backup file and must restore from an earlier backup. This scenario presents a risk of transaction logs that are out of sequence due to a “fork-in-time.” Consider the following sequence of repository events:

1. Generate backup1.
2. Generate transaction logs t1, t2, t3.
3. Generate backup2.
4. Generate transaction logs t4, t5, t6.
5. Restore backup2.
6. commitRestore (without replaying transaction logs t4, t5, t6).

The repository is now at same state as step 3.

7. Generate transaction logs t7, t8, t9.
8. Restore backup1.
9. Replay transaction logs t1 through t9.

In terms of the repository lifecycle, this scenario has two timelines, with a fork-in-time at the end of t3:



If, at step 5, we also restored the transaction logs (t4, t5, t6), 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 t1 through t6 without problems, but any changes made in (t7, t8, t9) would be lost.

During step 9, the replay of (t7, t8, t9) is likely to produce problems. If any object changes made in (t4, t5, t6) are logically inconsistent with those made in (t7, t8, t9),

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 (page 8-9). 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.

Methods for the restore process

To support the restore process, the following methods in class Repository (category Backup and Restore) are provided:

`restoreStatusOldestFileId`

This method returns the `fileId` of the oldest transaction log that needs to be present for the next restore from log operation, or `nil` if restore is not active.

`restoreToEndOfLog: fileId`

This method is used after a restore from an online extent backup or Smalltalk full backup, to restore all transaction logs up to and including the log with the given `fileId`. By default, the method reads log files from the directories specified by the Stone's `STN_TRAN_LOG_DIRECTORIES` configuration file parameter. If a `setArchiveLogDirectories:` or `restoreFromArchiveLogs` method has been executed since the previous `startstone` command, the method reads log files from those directories instead.

`setArchiveLogDirectories: arrayOfDirectorySpec`

This method specifies an Array of file system directories and/or raw devices to be used by subsequent invocation(s) of `restoreFromArchiveLogs` or `restoreToEndOfLog`. The filenames of the transaction logs are expected to be of the form `tranlogfileId.dbf`.

`setArchiveLogDirectory: aDirectorySpec`

This method specifies a single file system directory or raw device to be used by subsequent invocation(s) of `restoreFromArchiveLogs` or `restoreToEndOfLog`. The filenames of the transaction logs are expected to be of the form `tranlogfileId.dbf`.

`setArchiveLogDirectory: aDirectorySpec tranlogPrefix: aPrefix`

This method is similar to `setArchiveLogDirectory`: (above), except that the transaction log filenames begin with `aPrefix` rather than `tranlog`.

9.6 How to Make an Offline Extent Backup

To make a file system backup of the extents while GemStone is running, see page 9-3.

You cannot make a file system backup of a running GemStone system unless checkpoints are disabled. If you make an extent file backup while GemStone is running and checkpoints are enabled, that backup is likely to be unusable.

You can safely perform a file system backup of the extents while GemStone is shut down, provided that GemStone was shut down cleanly, such as by **stopstone** *gemStoneName*. 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 the necessity of backing up existing transaction logs.

To restore the repository using offline extent file backups, see the next section, "How to Restore from an Offline Extent Backup."

9.7 How to Restore from an Offline Extent Backup

Privileges required: FileControl.

This section describes how to restore the repository from an operating system backup made using **dump**, **tar**, **cp**, or similar methods. For complete recovery, this

backup must have been made while the repository monitor was shut down. That is, the backup must have been started after a **stopstone** and must have been completed before the subsequent **startstone**.

If the file system itself has been corrupted, not just the extent files, see the section “Disk Failure or File System Corruption” on page 4-20.

NOTE

Before you begin, make sure that you have operating system backups of good extents and that the backups were made after an orderly shutdown of the Stone repository monitor. If a backup consists of multiple files, the complete set must be available.

Step 1. If GemStone is still running, tell all users to log out and use **stopstone** to stop the repository monitor. Certain file system failures while the Stone is running may make it necessary to use **kill processid** to kill the Stone process.

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.

WARNING

Do NOT delete the transaction log files up to the time of the crash—leave them online in their current locations.

Step 3. Delete all extent files specified by `DBF_EXTENT_NAMES` in your configuration file.

Step 4. Restore 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 during recovery. 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. The next step depends on whether full transaction logging was in effect at the time the backup was made.

- If partial transaction logging (`STN_TRAN_FULL_LOGGING = False`) was in effect at the time the backup was made, the restore process is complete. Restart GemStone by invoking **startstone** in the usual manner.

- ❑ If full transaction logging (`STN_TRAN_FULL_LOGGING = True`) was in effect, continue by restoring from transaction logs. Use **startstone -R** to restart GemStone.

Continue with Step 7.

Step 7. Determine the location of all needed transaction logs. 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 03/02/07 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`.

The methods in category Configuration File Access of class System allow you to inspect the log directory paths and file name prefixes for the current configuration (for information, see “How to Access the Server Configuration at Run Time” on page 1-38).

Step 8. Restore the transaction logs in time sequence, beginning with the log identified by `restoreStatus`. How you do restore the logs depends on where the oldest logs are located and is described next. If you encounter a failure because of a truncated or corrupted transaction log, refer to “Errors While Restoring Transaction Logs” on page 9-26.

- ❑ If all transaction log files beginning with `tranlogfileId.dbf` are in the locations specified by the `STN_TRAN_LOG_DIRECTORIES` configuration option, skip to Step 9.
- ❑ If several of the older transaction logs have been moved to a different disk location, send `Repository>>setArchiveLogDirectories:` and `Repository>>restoreFromArchiveLogs` to restore those log files.

The following example restores logs archived in /GS-archive, which is not one of the active locations listed in STN_TRAN_LOG_DIRECTORIES:

```
topaz> login
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository setArchiveLogDirectories:
  #( 'GS-archive' )
SystemRepository restoreFromArchiveLogs
%
```

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.

Continue with the next substep if it applies, or else with Step 9.

- If individual transaction logs need to be restored, execute Repository>>restoreToEndOfLog: *fileOrDevice* for each file or raw partition in time sequence. (If the files are on tape, the files must first be restored to a disk drive, but they do not need to be in their original location.) For example:

```
topaz> login
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
  restored to 03/02/07 13:51:19 PST
  next fileId = 7, record = 10.
topaz 1> printit
SystemRepository restoreToEndOfLog: '%/tranlog7.dbf'
%
Restore from transaction log succeeded.
```

Continue with Step 9 to restore from online logs.

- Step 9.** After you restore transactions from any offline log files, restore transactions from the online 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> login
topazm: in login, found topaz session 1 stale
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository restoreFromCurrentLogs
%
Restore from transaction log(s) succeeded. [null]
```

Step 10. If restoration from the transaction logs was successful, send the message `commitRestore` to tell the system that you are finished restoring. No further logs can be restored, and normal user commits will be allowed.

```
topaz> login
topazm: in login, found topaz session 1 stale
gci login: currSession 1 rpc gem processId -1
successful login
topaz 1> printit
SystemRepository commitRestore
%
Restore from transaction log(s) succeeded. commitRestore
succeeded
```

Step 11. Make a new GemStone backup as soon as operational circumstances permit.

9.8 How to Recover After Repair of the File System

In case of a disk failure or a corrupt file system, the file system must be repaired before you can restart GemStone. The procedure you need to follow depends on how the damage was repaired.

To Recover After a File System Repair With `fsck`

After a repair with `fsck` (the UNIX file system consistency check and interactive repair utility), use `pageaudit` to check the condition of the system repository. (For instructions, see “How to Audit the Repository” on page 8-9.)

- If the page audit succeeds, try to restart GemStone again. If GemStone starts, you can resume normal operations.

- If the page audit fails or GemStone doesn't start, you will need to restore the repository file. (See "How to Restore from an Online Extent Backup" on page 9-7 or "How to Restore from a Smalltalk Full Backup" on page 9-14.)

To Recover When a File System Must Be Restored

If your system administrator intends to restore the file system from a backup device, it might be worthwhile to copy the repository to another node or to tape **before** restoring the file system. Although this copy may prove unusable, if a great deal of important data has been committed since the last backup, it may be worth a try.

To restart GemStone after the file system is restored, perform the following steps. (See Figure 9.2 on page 9-40.)

Step 1. If you made a copy of the repository, begin with that copy. To test the copy, use the methods discussed in the section "How to Audit the Repository" on page 8-9. You will need to specify the name and path of the copy using a temporary configuration file so that **pageaudit** is not performed on the extent that was restored along with the rest of the file system.

If you didn't make a copy of the repository or the copy does not pass **pageaudit**, start with the current extent file(s) that were restored from the file system backup. Review the log file and determine whether the backup was made while GemStone was running.

- If the Stone was running and checkpoints were not disabled and any changes were being made to the repository during the backup, the extent file(s) may be inconsistent and cannot be made to work. In that case, you must restore from a valid GemStone backup. However, the transaction logs should be usable.
- If the backup was done while GemStone checkpoints were suspended, continue to the next step.

Step 2. Do a **pageaudit** to check the current (restored) extent files. (For instructions, see "To Perform a Page Audit" on page 8-10.)

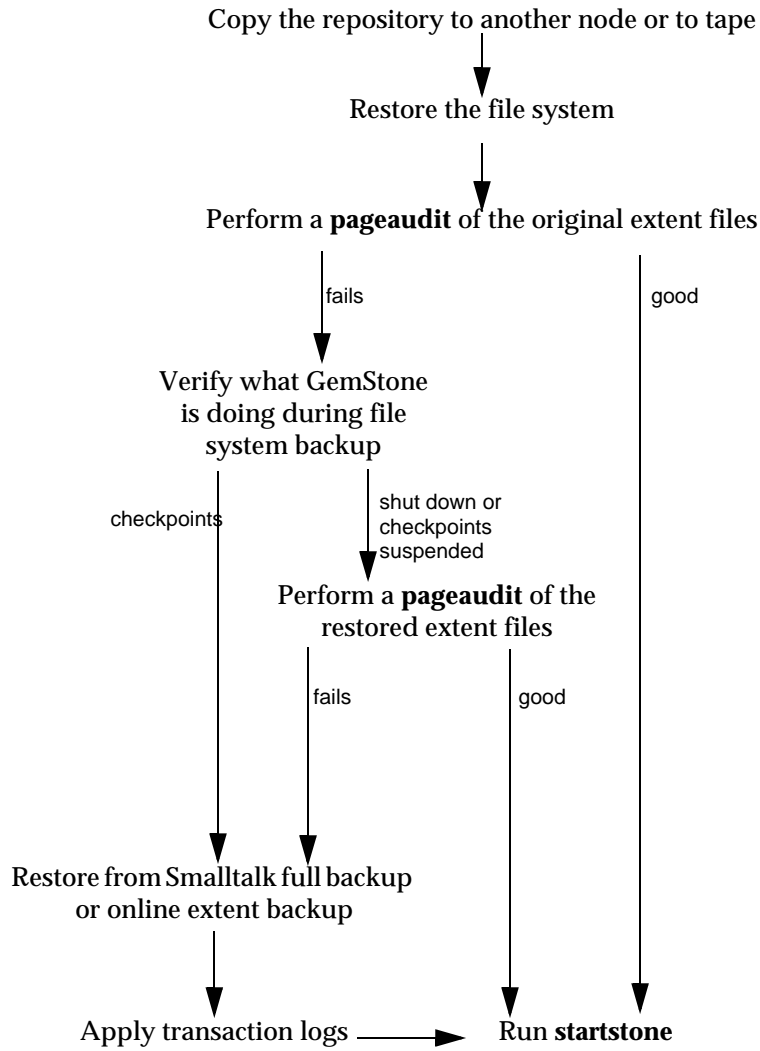
- If the page audit is good, try to restart the system again with **startstone**. If GemStone starts, you can resume normal operations.

- ❑ If the page audit fails or GemStone doesn't start, you will need to restore from a GemStone online extent backup (page 9-7) or from Smalltalk full backups (page 9-14).

NOTE

*Remember that **startstone** uses (1) a `GEMSTONE_SYS_CONF` environment variable or (2) `$GEMSTONE/data/system.conf` as the default system-wide configuration file. If you want it to use parameters from a different configuration file, be sure to specify that file with the **startstone -z** option. For a discussion of how **startstone** searches for the configuration file, see page A-2.*

Figure 9.2 Recovering When a File System Must Be Restored



9.9 How To Manage a Warm Standby System

To maximize system availability, you can run two GemStone installations, one as the primary system and the other on standby, ready to take over as quickly as possible in case of any failure of the primary system. To do so:

- Step 1.** Run the primary system using full logging as described in “Choosing a Logging Mode” on page 1-27.
- Step 2.** To protect against hardware failure, mirror its extents and transaction logs, as described in “Developing a Failover Strategy” on page 1-11.
- Step 3.** Make a Smalltalk full backup of the primary system. (Instructions start on page 9-8.) You’ll have to do this at least once, when you start this project; however, regular backups will simplify matters when you need to synchronize the primary and the standby systems.
- Step 4.** Run the standby system in restore mode. After each full backup of the primary system, restore the full backup to the standby system, leaving it in restore mode when you’re done. (Instructions start on page 9-16.)
- Step 5.** As each transaction log completes on the primary system, move the log to a file system accessible to the standby GemStone installation, if necessary, and replay the transaction log. (Instructions start on page 9-19.)
- Step 6.** If the primary system fails, replay its latest transaction log on the standby system. 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 applications so that, after detecting a failure, they can determine which system is the new primary and reconnect correctly.

- Step 7.** Correct the problem on the failed system and restart it.
- Step 8.** 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.

You can limit each transaction log to a tolerable amount of information either by limiting transaction log size or starting a new log at regular time intervals:

- **Size-based:** Limit the transaction log size, as described in “Choosing the Log Location and Size Limit” on page 1-29. When a transaction log grows to the specified limit, GemStone starts a new transaction log.
- **Time-based:** Run a script at regular intervals on your primary system that terminates the current transaction log and starts a new one.

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 a growing business, 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*.

GemStone implements a variety of garbage collection mechanisms, some automatic and some as a result of actions you take as you deem necessary. All these mechanisms can be tuned in various ways to best suit your operations.

This chapter describes GemStone's garbage collection mechanisms and explains how and when to use them.

This chapter discusses the following topics:

- **Basic Concepts** — the main concepts underlying garbage collection (page 10-2).
- **Automatic Garbage Collection** — garbage collection from local object memory (page 10-11), and epoch garbage collection (page 10-17).
- **Invoking Garbage Collection** — when, how, and why to invoke the garbage collection mechanisms that must be started by administrators (page 10-26).

10.1 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. 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 all the symbol lists for a given user, and those symbol 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 reclamation 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 10.1, a user creates a SymbolAssociation in the SymbolDictionary Published. The SymbolAssociation is an object (oop 70089) that refers to two other objects, its instance variables key (#Cost, oop 76859), and value (5.75, oop 70091).

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 10.1 An Association Is Created and Committed

```
topaz 1> display oops
topaz 1> printit
Published at: #Cost put: 5.75 .
Published associationAt: #Cost .
%
[70089 sz:2 cls: 873 SymbolAssociation] a SymbolAssociation
  key          [76859 sz:4 cls: 867 Symbol] Cost
  value       [70091 sz:8 cls: 761 Float] 5.7500000000000000E+00

topaz 1> commit
Successful commit
```

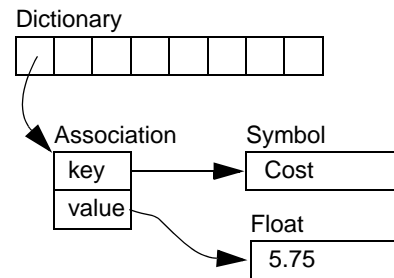


Figure 10.2 shows a second Topaz session that logs in at this point. Notice that the Topaz prompt identifies the session by displaying a digit. Because Session 1 committed the SymbolAssociation to the repository, Session 2 can see the SymbolAssociation.

Figure 10.2 A Second Session Can See the Association

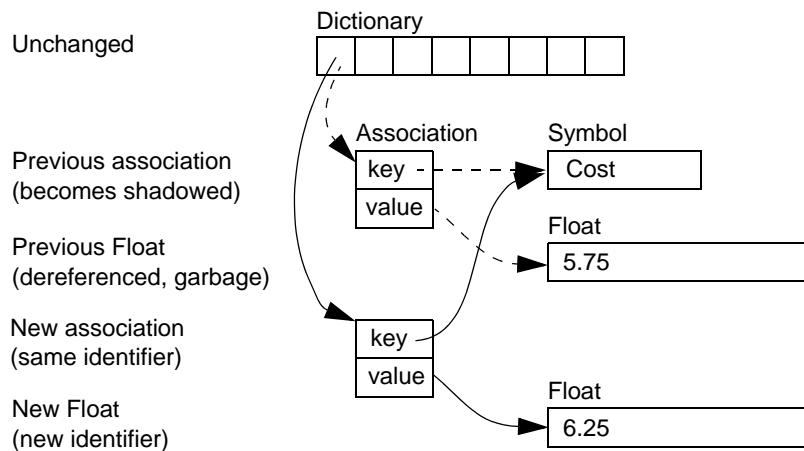
```
topaz 2> display oops
topaz 2> printit
Published associationAt: #Cost .
%
[70089 sz:2 cls: 873 SymbolAssociation] a SymbolAssociation
  key          [76859 sz:4 cls: 867 Symbol] Cost
  value       [70091 sz:8 cls: 761 Float] 5.7500000000000000E+00
```


Now Session 1 changes the *value* instance variable, creating a new SymbolAssociation (Figure 10.3). Notice in the oops display that the new SymbolAssociation object has the same identifier (70089) as the previous Association.

Figure 10.3 The Value Is Replaced, Changing the Association

```
topaz 1> printit
Cost := Cost + 0.50 .
Published associationAt: #Cost .
%
[70089 sz:2 cls: 873 SymbolAssociation] a SymbolAssociation
  key          [76859 sz:4 cls: 867 Symbol] Cost
  value        [70069 sz:8 cls: 761 Float] 6.2500000000000000E+00

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 70091) 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 10.4). 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 10.4 Session 2 Sees Change After Renewing Transaction View of Repository

```
topaz 2> printit
Published associationAt: #Cost .
%
[70089 sz:2 cls: 873 SymbolAssociation] a SymbolAssociation
  key          [76859 sz:4 cls: 867 Symbol] Cost
  value       [70091 sz:8 cls: 761 Float] 5.7500000000000000E+00

topaz 2> abort
topaz 2> printit
Published associationAt: #Cost .
%
[70089 sz:2 cls: 873 SymbolAssociation] a SymbolAssociation
  key          [76859 sz:4 cls: 867 Symbol] Cost
  value       [70069 sz:8 cls: 761 Float] 6.2500000000000000E+00
```

Only when all sessions with concurrent transactions have committed or aborted can the shadow object be garbage collected.

What Happens to Garbage?

GemStone implements a variety of garbage collection mechanisms tailored for specific scenarios. The following idealized process describes none of them. Instead, it provides a general framework you can use to understand how each different mechanism handles the scenarios it was designed to handle.

NOTE

Not every garbage collection mechanism uses every step of the process below. Subsequent sections describe each specific process in detail.

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 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 halts 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 GcGem*, 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. One or more *Reclaim GcGems* 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 GcGem commits. The reclaimed OOPs are returned to their free pool.

9. The Reclaim GcGem's commit record is disposed of. The reclaimed pages are returned to their free pool.

Admin and Reclaim GcGems

It is useful to understand the distinction between the Admin GcGem and Reclaim GcGems:

- The Admin GcGem finalizes the vote on possibly dead objects (page 10-7, Step 5). The Admin GcGem handles compiled methods and other special cases.

The Admin GcGem also performs epoch garbage collection. For details, see "Epoch Garbage Collection" on page 10-17.

The Admin GcGem is started by the Stone at startup time if enabled by the `STN_ADMIN_GC_SESSION_ENABLED` configuration option. For details, see "Running the Admin GcGem" on page 10-29.

- Reclaim GcGems are dedicated to the task of reclaiming shadowed pages and dead objects repository-wide, along with their OOPs. If you have multiple extents in your repository, you can run multiple Reclaim GcGems, up to one on each extent file in the repository. Each Reclaim GcGem can handle several extents.

The configuration parameter `STN_NUM_GC_RECLAIM_SESSIONS` specifies the maximum number of Reclaim GcGems that can be started. For details, see "Configuring and Starting the Reclaim GcGems" on page 10-36.

By default, one Admin GcGem and one Reclaim GcGem are configured to run in the configuration file, and are started automatically when the Stone is started. This is the simplest GcGem configuration.

- We recommend that you leave the Admin GcGem running at all times, although this GcGem is required only following a `markForCollection` or `markGcCandidatesFromFile:` garbage collection operation. (Subsequent sections of this chapter describe these operations in detail.) If the Admin GcGem 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 at least one Reclaim GcGem running at all times.

Both the Admin and Reclaim GcGems are run from the GcUser account, a special account that logs in to the repository to perform garbage collection tasks. For more about this account, see Chapter 5, "User Accounts and Security".

Different Ways to Collect 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.

Where

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.

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 reclamation of the page on which the object resides.

How

Some garbage collection mechanisms, such as the process that scavenges local object memory, occur automatically when needed in a manner invisible to the user. As a unit, these automatic mechanisms are designed to catch as much garbage as possible. But they can't catch everything; therefore, the system administrator has a variety of additional tools: Smalltalk methods to invoke a wide variety of other mechanisms. You can find the right subset of these to run at intervals appropriate for your operational profile.

GemStone's Garbage Collection Mechanisms

GemStone provides the following mechanisms that together mark and reclaim garbage, thereby helping you to control repository growth.

Both Marking and Reclaiming

Temporary object memory collection — This mechanism, which you might think of as preventive, reclaims objects in local object memory while they exist within view of only the session that created them. Each Gem session process performs this task automatically. The goal is to prevent temporary garbage objects from reaching

permanent object memory through a committed transaction. These objects are collected individually, instead of in pages.

You can set GemStone environment variables and configuration options to manage the use of temporary object memory. For details, see “Temporary Object Memory Garbage Collection” on page 10-11.

Marking Only

Repository-wide marking — To prevent the repository from growing large enough to cause problems on a regular basis, you can run `Repository >> markForCollection` (page 10-26). This method combines a full sweep of all objects in the repository and the marking of each possible dead object in a single operation.

FDC/MGC — This two-step process allows for more efficient garbage collection in larger production systems. In the first step (FDC), the method `Repository >> findDisconnectedObjectsAndWriteToFile`: explicitly finds disconnected objects in the repository and identifies a list of possible targets for garbage collection. Because the FDC method writes the possibly dead objects to a file, you can run this step in a copy of the repository rather than in the production system. In the second step (MGC), the method `Repository >> markGcCandidatesFromFile`: examines the set of candidate garbage objects, ensures that they are not referenced by live objects, and then marks them as dead for subsequent garbage collection. For details about the FDC/MGC process, see page 10-31.

Epoch garbage collection — Periodically, the Admin GcGem 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. For details about epoch garbage collection, see page 10-17.

Reclaiming Only

Parallel reclamation — Once you’ve run `markForCollection`, FDC/MGC, or epoch garbage collection, you can run `Reclaim GcGems` to reclaim pages that contain either dead or shadow objects. When there are a high number of objects needing to be reclaimed, you can run multiple `Reclaim GcGems` simultaneously, as many as one `Reclaim GcGem` for each extent in your repository. For details about reclaiming pages, see page 10-35.

10.2 Temporary Object Memory Garbage Collection

Collecting garbage from local object memory is an automatic process.

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 `GsMethod` 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.

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.

Examining and Allocating 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.

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.

Environment Variables

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_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 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.

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.

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 A-19.

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_INITIAL_SIZE

On HP-UX and AIX, the initial size (in KB) of temporary object memory that is allocated on startup. As temporary object memory fills up, the amount of temporary memory available is increased to the limit of **GEM_TEMPOBJ_CACHE_SIZE**. This setting is not used on other platforms, where memory is handled differently.

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:

- Default 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 (page 10-12) to produce the resulting printouts. The examples shown are for Solaris and may vary on other platforms.

Default Configuration

A default value (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
```

How Memory Is Allocated

On Solaris and Linux, the memory spaces configured by `GEM_TEMPOBJ_CACHE_SIZE` and `GEM_TEMPOBJ_POMGEN_SIZE` are reserved in virtual memory at Gem startup, and are allocated/released as needed via `mmap()` calls to Solaris or Linux. On HP-UX and AIX, memory is handled differently, and may not be reserved without being allocated. On these platforms, the memory configured by `GEM_TEMPOBJ_INITIAL_SIZE` is initially allocated. As more memory is needed, a new memory region is allocated with `mmap()`, contents of previous memory is copied to the new memory, and the previous memory is released with `munmap()`.

This means that you can configure `GEM_TEMPOBJ_CACHE_SIZE` large enough to handle the worst case of infrequent transactions with many temporary or modified-committed objects. It is OK to have `maxSessions*(2.5 * GEM_TEMPOBJ_CACHE_SIZE)` exceed the memory budget for sessions if the average memory usage is within the physical memory budget.

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 `#rtErrSignalAlmostOutOfMemory`, 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.

For more information on handling the `#rtErrSignalAlmostOutOfMemory`, see the *Programming Guide for GemStone/S 64 Bit*.

10.3 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 `GcGem` 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 parameter to `TRUE`. For details about this parameter, see page A-27.
- Execute the method `System Class >> enableEpochGc`. See “Methods for Epoch Garbage Collection” on page 10-18.

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. An in-depth discussion of the performance tradeoffs of short or long epochs starts on page 10-20.

To change the mark/sweep buffer size of the epoch garbage collection, adjust the `GcUser` configuration parameter `#epochGcPageBufferSize`. The default value is 150 pages.

Methods for Epoch Garbage Collection

Privileges required: `GarbageCollection`.

Class `System` (category `Transaction Control`) provides the following class methods that you can use to control epoch garbage collection:

`disableEpochGc`

Disables epoch garbage collection, and appends `STN_EPOCH_GC_ENABLED=false` to the Stone’s configuration file. This method has no effect if epoch garbage collection is already disabled.

After this method is successfully executed, no further epoch garbage collection operations will be run, but an epoch garbage collection operation already in progress when this method is executed will not be interrupted.

`enableEpochGc`

Enables epoch garbage collection, and appends `STN_EPOCH_GC_ENABLED=true` to the Stone's configuration file. This method has no effect if epoch garbage collection is already enabled.

`forceEpochGc`

If epoch garbage collection is enabled, force an epoch garbage collection to run as soon as possible, regardless of the setting of the `GcUser` configuration parameters `#epochGcTimeLimit` or `#epochGcTransLimit`. Returns true if the epoch garbage collection was started, false if not.

If successful, this method sets the Stone cache statistic `EpochForceGc` to 1. Once the epoch garbage collection has started, `EpochForceGc` returns to 0.

If any of the following conditions exists, this method fails and returns false:

- 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 GC session 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.

`clearEpochGcState`

Resets the epoch GC state. The epoch GC state keeps track of objects eligible to be marked as possible dead during the next epoch garbage collection. This method has no effect on objects already marked possible dead by previously completed epoch garbage collection operations.

Cache Statistics

Several cache statistics include information about the epoch garbage collection process. The best way to view this data is to start the `statmonitor` utility and then

use VSD (the visual statistics display tool) to view the data. For details about statmonitor and VSD, see Appendix F.

You may find it useful to examine the following statistics:

EpochGcCount

The number of times that the epoch garbage collection process was run by the GcGem since the GcGem was started. For a system in steady state, look for uniform periods between runs or a uniform run rate.

EpochNewKobjs

The number of new objects that were created during the last epoch.

EpochPossibleDeadKobjs The number of possible dead objects found by the last epoch garbage collection.

EpochScannedKobjs

The number of objects scanned by the last epoch garbage collection.

For more information about these and other GemStone statistics, see “Cache Statistics” on page 8-23.

Information is also available by executing the method `System class>>cacheStatistics:` for the Stone’s cache slot.

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`
The maximum frequency of epoch garbage collection—by default, 60 minutes (3600 seconds). We recommend setting it to at least 30 minutes.
- `#epochGcTransLimit`
The number of transactions required to trigger epoch garbage collection—by default, 5000.

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 10.5 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 10.5 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.

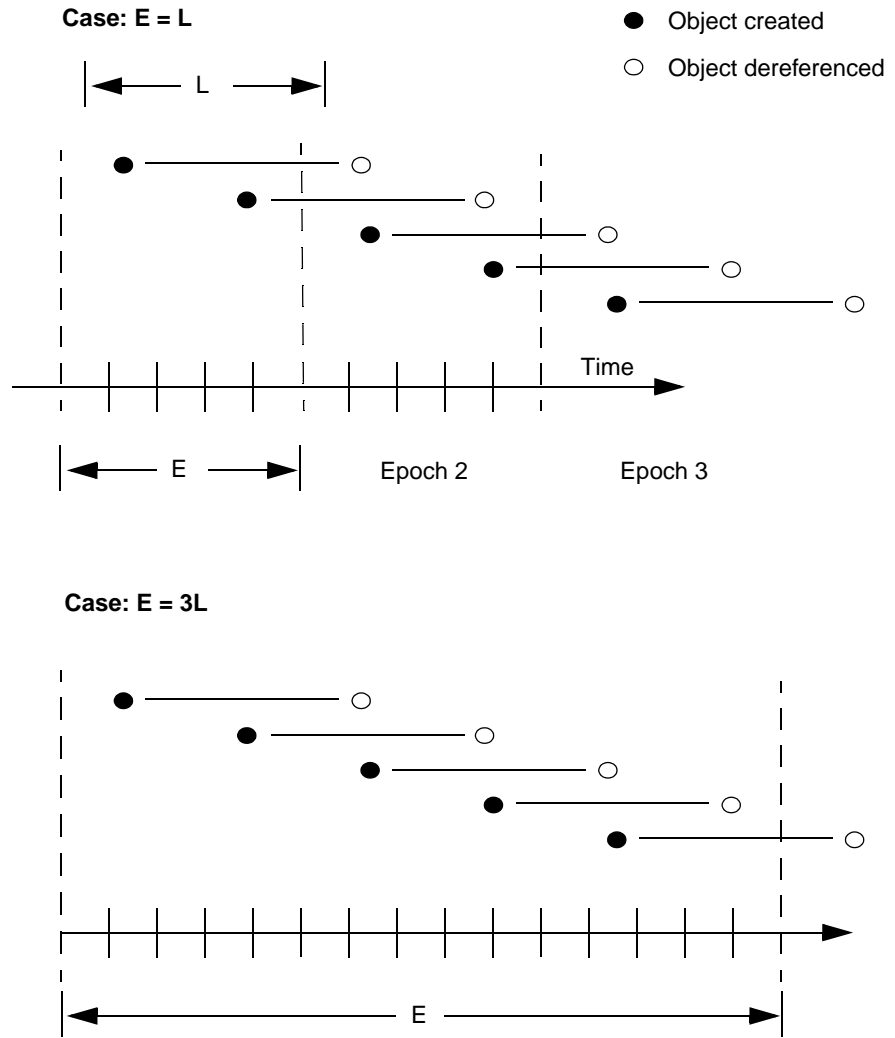
The results shown in Figure 10.5 can be expressed as:

$$\begin{aligned}\text{Objects Missed by EpochGC} &= R \times L \\ \text{Objects Recovered by EpochGC} &= R(E - L)\end{aligned}$$

For example, assume $R = 1000$ objects per minute, $L = 5$ minutes, and $E = 15$ minutes. Then, for each epoch:

$$\begin{aligned}\text{Objects Missed} &= 1000 \times 5 = 5000 \\ \text{Objects Recovered} &= 1000 (15 - 5) = 10000\end{aligned}$$

Figure 10.5 Effect of Collection Interval on Epoch Garbage Collection



Therefore:

- Set #epochGcTimeLimit $E >$ lifetime L of short-lived objects.

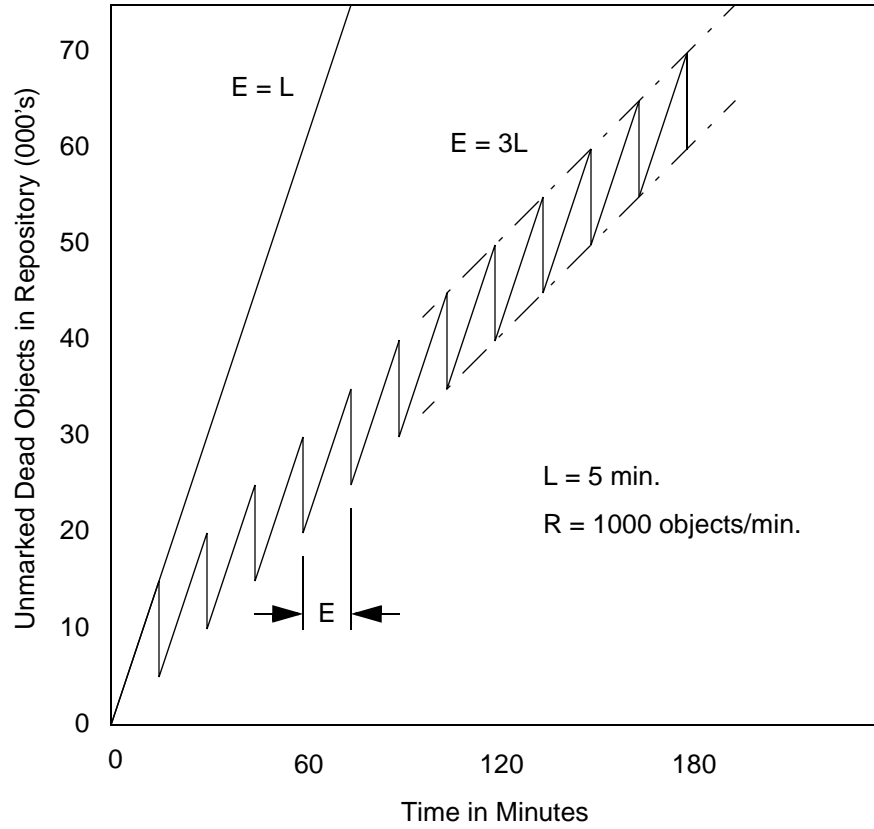
Figure 10.6 graphs the effect of the epoch. When $E = L$, epoch garbage collection is in effect disabled—all objects survive into the next epoch; the number of unmarked yet dead objects in the repository grows at the creation rate. These dead objects remain unidentified until you run `markForCollection`.

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:

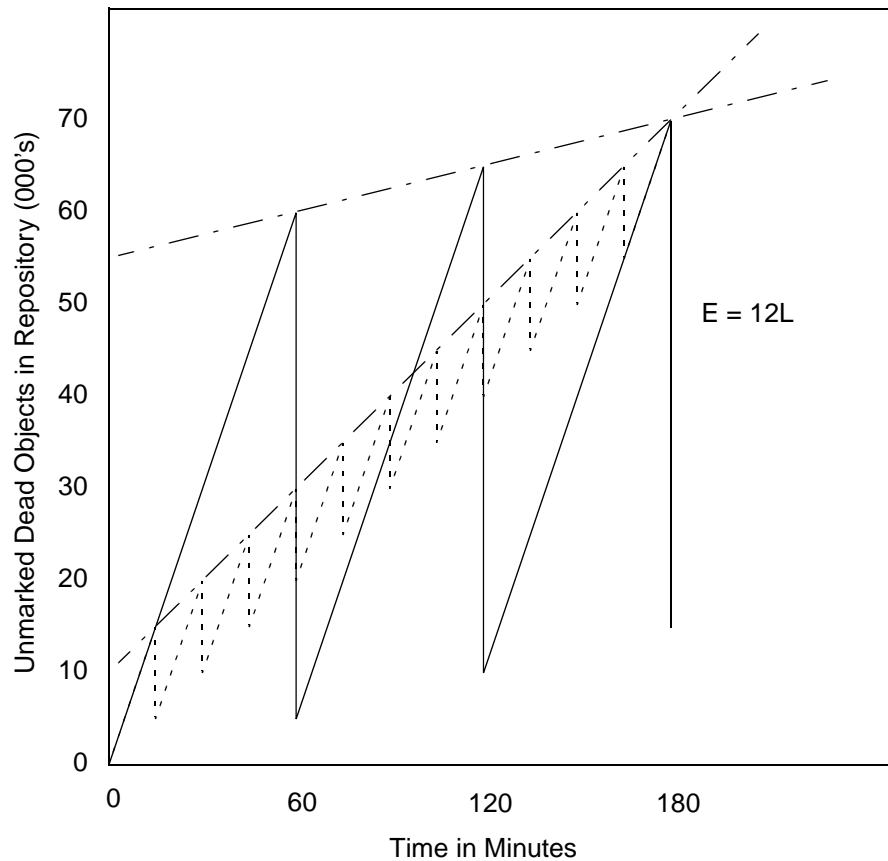
- moderate growth in the repository, and
- the need to run `markForCollection` often enough to mark dead objects that survive between epochs.

Figure 10.6 Repository Growth with Short Epoch



Suppose we extend the epoch to $E = 12L$. The result is shown in Figure 10.7, superimposed on part of the previous figure.

Figure 10.7 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.

10.4 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 10-2. 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 10-6. It does not reclaim the space or OOPs itself—the `Reclaim GcGems` do that, as described beginning on page 10-35.

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 “Reducing Impact on Other Sessions” on page 10-28.

When `markForCollection` completes successfully, the `Gem` that started it displays a message such as the one below:

```
Successful completion of markForCollection.
    2694458 live objects found.
    129654 possible dead objects, occupying approximately
    11668860 bytes, may be reclaimed.
```

This method aborts the current transaction and runs `markForCollection` outside of a transaction so that the session doesn't interfere with ongoing reclamation. When `markForCollection` completes, the session reenters a transaction, if it was in one when this method was invoked.

Because it runs outside of a transaction, `markForCollection` must respond to `#rtErrSignalAbort` messages from the Stone. To avoid excessive interference with the marking process, consider temporarily raising the `#StnSignalAbortCrBacklog` internal parameter (page A-35) to let `markForCollection` run for about five minutes between such signals; the precise value necessary depends on the your application's commit rate.

Only SystemUser can change the #StnSignalAbortCrBacklog parameter. For information about setting internal parameters, see “To Change Settings at Run Time” on page 1-39.

If another garbage collection task is in progress at the time you try to do markForCollection, this method may report a conflict error similar to that shown below. (Before issuing the error, the markForCollection method waits up to two minutes for the other garbage collection to finish.)

```
GemStone: Error           Nonfatal  
Garbage collection aborted, reason: 'gcLock denied, vote  
state 1 =  
voting, session not voted: sessionId 5 pid 21336 on  
pelican', conflict  
code: --- . Garbage collection was aborted and should be  
tried again later.  
Error Category: [GemStone] Number: 3006 Arg Count: 1  
Arg 1: gcLock denied, vote state 1 = voting, session not  
voted: sessionId 5 pid 21336 on pelican
```

This concurrency conflict may be caused by one of these conditions:

- A markGcCandidatesFromFile: or markForCollection is in progress in another session.
- A previous markGcCandidatesFromFile: or markForCollection completed the mark phase, but voting on possibly dead objects has not completed.

NOTE

For voting to complete, the Admin GcGem must be running, as described on page 10-29. Also, long-running sessions that are idle and never abort will prevent the vote from completing.

Before issuing the error, the markForCollection method waits up to two minutes for the other garbage collection to finish. You can try markForCollection again after a few minutes or use markForCollectionWait: as described below.

To have the markForCollection wait longer than two minutes for another garbage collection to complete, 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 appear to 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 GcGem is running to complete processing of dead objects once the vote is completed. (See “Running the Admin GcGem” on page 10-29.)

Reducing Impact on Other Sessions

Because `markForCollection` can make extensive use of system resources for a long time, you may need to reduce its impact on other sessions.

To increase the speed of garbage collection, you can increase the size of the mark/sweep buffer for the user running `markForCollection`—typically `GcUser` or `DataCurator`. To do so, log in as the appropriate user and evaluate:

```
UserGlobals at: #mfcGcPageBufSize put: newValue
```

A value of 1000 while `markForCollection` runs is a good place to start. (The default value is 320.)

NOTE

A `sigAbort` causes `markForCollection` to redo its recent work, which in the worst case can include all work done since the last `sigAbort`. Therefore, in an environment where the `markForCollection` session gets `sigAborts`, increasing the size of the mark/sweep buffer can degrade session response time to `sigAborts`, leading to commit record backlog problems and potential lostOTRoot failures of the `markForCollection` session, as well as significantly degrading `markForCollection` performance. In some cases, `markForCollection` effectively hangs for as long as the process continues to get `sigAborts`.

You'll need to experiment to determine the best value for `mfcGcPageBufSize`, given your operational profile, but values above 6000 are seldom required and usually detrimental.

Additional impact on other sessions may occur after the `markForCollection` has completed because the results may cause an increase in the number of pages that are candidates for reclaiming.

Scheduling markForCollection

To invoke `markForCollection` using the `cron` facility, create a three-line script file similar to the Topaz example on page 10-26 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.

10.5 Running the Admin GcGem

Privileges required: `GarbageCollection`

The Admin GcGem must be running to complete processing of dead objects following a `markForCollection` or `markGcCandidatesFromFile:` operation. If the Admin GcGem is not running following one of these operations, the garbage collection process cannot complete, and garbage can build up in your system.

Before you can start the Admin GcGem, you must configure it to run. You can do this either by editing the configuration file (by default, `system.conf`) before starting the Stone or by using runtime methods to set the configuration. If you set the configuration during runtime, your changes are lost when the Stone repository monitor is subsequently restarted. For permanent changes, edit the configuration file.

Editing the Configuration File

The configuration parameter `STN_ADMIN_GC_SESSION_ENABLED` (page A-24) specifies whether the Admin GcGem is started by the Stone at startup time.

Configuring the Admin GcGem at Run Time

Before taking any action that requires the presence or absence of the Admin GcGem, you should verify whether the Admin GcGem is configured to run:

```
System configurationAt: #StnAdminGcSessionEnabled
```

If this method returns 1, the Admin GcGem is configured to run. If not, configure the Admin GcGem by executing:

```
System configurationAt: #StnAdminGcSessionEnabled put: 1.
```

Configuring the Admin GcGem to be enabled does not start it; starting the Admin GcGem requires a separate step.

Starting the Admin GcGem

To start the Admin GcGem (which you have already configured to run):

```
System startAdminGcSession
```

Alternatively, you can start the Admin GcGem together with all Reclaim GcGem sessions that are configured to run:

```
System startAllGcSessions
```

Both `startAdminGcSession` and `startAllGcSessions` initiate the start of the GcGems, and return immediately. The GcGems may take longer to actually initialize and start up.

To wait a given period of time for the Admin GcGem and all Reclaim GcGems to start up:

```
System waitForAllGcGemsToStartForUpToSeconds: anInt
```

This method starts all the GcGems that are configured, and waits for the specified number of seconds. If all the GcGems have not started up within that time, the method returns 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 but all GcGems eventually start correctly.

To verify that the Admin GcGem is running:

```
System adminGcGemSessionId
```

This returns a `SmallInteger` representing the session ID of the Admin GcGem, or 0 if the Admin GcGem is not running.

To verify that the Admin GcGem is running, and that each extent has a Reclaim GcGem started:

```
System hasMissingGcGems
```

This returns false if the Admin GcGem is running and each extent has been assigned a Reclaim GcGem.

Stopping the Admin GcGem

To stop the Admin GcGem:

```
System stopAdminGcSession
```

To configure the Admin GcGem to not run:

```
System configurationAt: #StnAdminGcSessionEnabled put: 0.
```

This also stops the Admin GcGem, if it is running.

To stop the Admin GcGem and any Reclaim GcGem sessions that are running, use:

```
System stopAllGcSessions
```

10.6 The FDC/MGC Process

As discussed in the previous section, `markForCollection` combines a full sweep of all objects in the repository and marking of each possible dead object in a single operation. In a large production system, this can have a significant effect on performance. To lessen this impact, you can perform garbage collection as a two-step process: `findDisconnectedObjectsAndWriteToFile:` (FDC) and `markGcCandidatesFromFile:` (MGC). These two steps together perform the same function as running `markForCollection`.

In this two-step process, the possibly dead objects are first written to a file, thus allowing you to run the FDC in a copy of the repository rather than in the production system. The OOPs of the possibly dead objects are then transferred to the production system, where they are marked as dead and subsequently garbage-collected.

Running in a copy of the repository avoids the performance impact of this processing on the production repository, and permits operations on the repository that would not be possible during a long-running repository sweep; for example, the production repository can be shut down.

- If you choose to run garbage collection in the production system rather than in a copy of the repository, `markForCollection` would be a better option.

Step 1. Prepare for FDC.

Performance of FDC is much faster if there are no shadowed objects. Before starting the FDC/MGC process on the production server, run enough Reclaim GcGems (page 10-35) to process all shadow objects, and make sure the commit record backlog is low.

FDC produces a large file of encoded OOPs; the size of the file (in bytes) is approximately four times the number of dead objects. On the server that will run the backup, ensure that there is enough space for the resulting file, and that the filename you will use does not already exist.

Step 2. Create a copy of the production repository.

Make a backup of your production repository. You may either make an online extent file backup (page 9-3), a Smalltalk full backup (page 9-8), or an offline extent file backup (page 9-33).

Restore the backup into another location. This is the repository that will run the FDC, while your production repository continues to operate normally.

Step 3. Run `findDisconnectedObjectsAndWriteToFile:...`

On the running backup repository, execute the FDC method:

```
SystemRepository
  findDisconnectedObjectsAndWriteToFile: filename.txt
  pageBufferSize: 640
  saveToRepository: true
```

This method does the following:

- Runs code similar to the `markForCollection` sweep phase on the backup repository.
- Stores the resulting list of candidate dead objects to the named file (*filename.txt*). The file is created in the current working directory, and contains the encoded OOPs for all of the candidate dead objects.
- If `saveToRepository:` is `True`, stores the list of encoded OOPs to an internal array as well. Set this argument to `True` if you want to be able to subsequently examine and analyze the results of the FDC operation; it is not needed as part of the garbage collection process.

If `saveToRepository:` is `True`, you can subsequently use the array of dead objects to recreate the FDC file in the event that the file is deleted or lost. See the following discussion, “If You Need to Recreate the FDC File.”

Also note that a larger `pageBufferSize` may improve performance when you run FDC offline in a backup repository, as illustrated here.

Step 4. At this point, you can shut down the backup repository; it is no longer needed for the MGC/FDC process.

Step 5. Run `markGcCandidatesFromFile:` (page 10-34) on the production system.

If You Need to Recreate the FDC File

If your FDC file should inadvertently be deleted or become lost, you may still be able to recreate the FDC file for subsequent use by `markGcCandidatesFromFile: (MGC)` — provided that `saveToRepository: was True` when you ran FDC.

You can execute this method to write the array of candidate dead objects found by the last FDC operation (for which `saveToRepository: was True`) to a file:

```
SystemRepository writeFdcArrayToFile: aFileNameString
```

NOTE

Before running this method, ensure that you have sufficient disk space to hold the entire file, whose size in bytes will be approximately four times the size of the array.

This method expects Globals at: `#FdcResults` to contain the array of candidate dead objects, in order and sorted by OOP. You must specify the file as a String, using the full path and filename of a file that does not yet exist.

The method returns true if successful, false if the entire file could not be written, or nil if the results of the last FDC could not be found.

Fast FDC

You can also run FDC in an alternate mode, which runs as quickly as possible, without regard to the impact on system resources. This mode assumes it is the only process running and can use as much of the cache as it needs.

To run fast FDC, use the following:

```
Repository findDisconnectedObjectsAndWriteToFile: filename.txt  
  pageBufferSize: pagebuf  
  saveToRepository: saveToRep  
  numCacheWarmers: numWarmers
```

When `numWarmers` is greater than zero, the fast FDC algorithm is used and garbage collection cache warmer gems are automatically started. If `numWarmers` is set to zero, then the regular FDC algorithm is used.

In fast FDC, Garbage Collection (GC) cache warmers are started during the FDC run. These special sessions do not have a view of the repository; they merely read data pages into the shared page cache before they are needed by the FDC gem. GC cache warmers are automatically stopped when the FDC gem logs out. Note that these GC cache warmers are distinct from Object table cache warmers.

Additional GC cache warmer gems may be started once the FDC has begun, by executing:

```
System _startGcCacheWarmer
```

While the fast FDC process will require tuning for optimal performance, the following are suggested as starting points for a 100GB repository:

- page Buffer size of 2000 to 5000.
- 10 GC cache warmer gems
- 2-4 free frame page servers
- SPC Object table cache warming completed (see the utility command “startcachewarmer” on page B-11).
- A disk architecture optimizing fast sequential reads with read ahead.

Run markGcCandidatesFromFile:

Privileges required: GarbageCollection.

`markGcCandidatesFromFile`: (MGC) performs an optimized linear sweep of the repository to identify any objects in the FDC output file that still have references. If the only references to candidate objects are from other candidates, the objects are marked for subsequent reclamation.

The MGC operation scans all *non-candidate* objects to ensure that there are no references to candidate objects. If there are no such references, the MGC operation is complete. If, however, there are any references to candidate objects, the MGC operation performs a transitive closure on all *former candidate* objects to identify any other candidate objects that cannot be collected.

To minimize the impact on performance, run the MGC operation at a time when the load on the production repository is low, such as during off-hours.

On the production system, execute:

```
SystemRepository markGcCandidatesFromFile: aFileNameString
```

where *aFileNameString* is a String representing the full path to the file of encoded OOPs that was created during the FDC (page 10-32, Step 3). Make sure that this file is available from the production location.

If another garbage collection (`markForCollection`) is in progress at the time you try to do `markGcCandidatesFromFile`: , or if voting has not completed, it

may report a conflict error similar to that shown below. Try `markGcCandidatesFromFile:` again after a few minutes.

The Garbage Collection process detected a concurrency conflict, reason:
#Garbage collection in progress by another session, try again later.

Recall that all marking only provides a set of possible dead objects for voting and eventual reclaiming, as described under “What Happens to Garbage?” on page 10-6. Marking does not by itself reclaim space or identifiers—the Reclaim GcGems do that, as described beginning on page 10-35.

As mentioned elsewhere in this chapter, the Admin GcGem must be running to complete processing of dead objects following an MGC operation. For details, see “Running the Admin GcGem” on page 10-29.

In some cases, there could be irrelevant errors that would prevent MGC from completing. To override such errors, execute the following method:

```
SystemRepository markGcCandidatesFromFile: aFileNameString  
forceOnError: aBoolean
```

This method is virtually identical to `markGcCandidatesFromFile:`, except that if *aBoolean* is true, the MGC will run even if the given file contains objects that fail validation, or there are references from live objects to some objects in the candidate collection. The live objects in the candidate collection are not marked as dead.

10.7 Reclaiming Pages

When a Reclaim GcGem is running, it examines pages marked reclaimable because they contain either dead or shadow objects. It also reclaims fragments of space left by transactions that did not fill an entire page.

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.

Two actions can hasten this moment:

- If your session is the only one logged in (other than GcGems), you can invoke page reclamation directly. See the following discussion, “Configuring and Starting the Reclaim GcGems.”

- If other users are logged in, you can determine which sessions are viewing the oldest commit record, thereby impeding reclamation. See the discussion, “To Identify Sessions Holding Up Page Reclamation,” on page 10-41.

Configuring and Starting the Reclaim GcGems

Privileges required: `GarbageCollection`

Before you can start one or more Reclaim GcGems, you must configure the repository to run the desired number of GcGems. You can do this either by editing the configuration file (by default, `system.conf`) before starting the Stone or by using runtime methods to set the configuration. If you set the configuration during runtime, your changes are lost when the Stone repository monitor is subsequently restarted. For permanent changes, edit the configuration file.

Editing the Configuration File

The configuration parameter `STN_NUM_GC_RECLAIM_SESSIONS` (page A-32) specifies the number of Reclaim GcGems that are started by the Stone at startup time. You may not configure more Reclaim GcGems than the number of extents in the repository, as specified in `DBF_EXTENT_NAMES` (page A-12). If you specify a value larger than the number of extents, one GC reclaim will be started for each extent.

Configuring Reclaim GcGems at Runtime

Before taking any action that requires the presence or absence of Reclaim GcGems, you should verify the number of Reclaim GcGems for which your repository is currently configured:

```
System configurationAt: #StnNumGcReclaimSessions
```

This method returns the number of Reclaim GcGems that are currently configured to run. These Reclaim GcGems may or may not be running.

When there are a high number of pages needing to be reclaimed, you might choose to shut down the single Reclaim GcGem during off-hours, and then start a larger number of Reclaim GcGems.

To update the number of Reclaim GcGems that are configured to run:

```
System configurationAt: #StnNumGcReclaimSessions put: N.
```

where *N* is the maximum number of reclaim GcGems that you intend to run. This method does not start the Reclaim GcGems.

To configure your system for the maximum number of Reclaim GcGems (one Reclaim GcGem per extent):

```
System configurationAt: #StNumGcReclaimSessions put:  
    SystemRepository numberOfExtents.  
System startAllReclaimGcSessions.
```

Since the GcGems place some load on heavily loaded systems (reclaimed pages must be committed), the performance impact of so many Reclaim GcGems may offset any potential advantage. Even when run during off-hours, a large number of Reclaim GcGems may cause commit bottlenecks for each other.

At least one Reclaim GcGem should be running at all times. In general, we recommend running one Reclaim GcGem for between 5 and 10 extents. You may need to experiment to find the correct balance for your system.

Configuring a Reclaim GcGem to be enabled does not start it; starting the Reclaim GcGem requires a separate step.

Starting Reclaim GcGems

To start all Reclaim GcGems that you have configured to run, execute one of the following methods:

```
System startAllReclaimGcSessions
```

This method starts the number of Reclaim GcGems that you have configured to run, dividing the extents evenly among them.

```
System startAllGcSessions
```

This method is similar to `startAllReclaimGcSessions`, but also starts the Admin GcGem.

Alternatively, you can explicitly assign Reclaim GcGems to a contiguous list of extents, as described in the next section, “Running Reclaim GcGems on a Range of Extents.”

All of these methods initiate the start of the GcGems, and return immediately. However, the GcGems may take longer to actually initialize and start up.

To wait a given period of time for all the GcGems (Reclaim and Admin) to start up:

```
System waitForAllGcGemsToStartForUpToSeconds: anInt
```

This method starts all the GcGems that are configured, and waits for the specified number of seconds. If all the GcGems have not started up within that time, the method returns 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` but all GcGems eventually start correctly.

Running Reclaim GcGems on a Range of Extents

A Reclaim GcGem can reclaim from a contiguous list of extents. You can control the allocation of Reclaim GcGems to extents using the method:

```
System startReclaimGemForExtentRange: startExtent to:
endExtent.
```

For example, if there are four extents in the repository, and you have configured to run two Reclaim GcGems, to start one Reclaim GcGem on the first extent, and the other on the other three extents, the code would look like this:

```
topaz 1> printit
System startReclaimGemForExtentRange: 1 to: 1.
System startReclaimGemForExtentRange: 2 to: 4.
%
true
topaz 1> printit
System currentGcReclaimSessionsByExtent
%
an Array
#1 3
#2 6
#3 6
#4 6
```

As shown here, the method `currentGcReclaimSessionsByExtent` returns an array whose size is the number of extents in the repository (in this example, four). The element at each index is the session ID of the Reclaim GcGem for that extent.

Similarly, for a repository with 20 extents, you could set up four Reclaim GcGems as follows:

```
topaz 1> printit
System startReclaimGemForExtentRange: 1 to: 5.
System startReclaimGemForExtentRange: 6 to: 10.
System startReclaimGemForExtentRange: 11 to: 15.
System startReclaimGemForExtentRange: 16 to: 20.
%
true
```

Running Reclaim GcGems on Remote Nodes

You can configure Reclaim GcGems to run on remote nodes (nodes other than the one the Stone is running on). To do this, use the following method:

```
System Class >> startReclaimGemForExtentRange: startExtent
to: endExtent onHost: hostNameOrIpAddress
```

For example, using the above example if the Reclaim GcGems should be run on a different node, the code might look as follows:

```
topaz 1> printit
System startReclaimGemForExtentRange: 1 to: 1 onHost:
'denile'.
System startReclaimGemForExtentRange: 2 to: 4 onHost:
'denile'.
%
true
topaz 1> printit
System currentGcReclaimSessionsByExtent
%
an Array
#1 3
#2 6
#3 6
#4 6
```

This code starts a remote shared page cache and page server on the remote machine, if they do not already exist. A GemStone NetLDI must be running on the remote node (for details, see page 3-4). Log files are written to the home directory of the user who owns the processes.

When the Stone is run on a multi-homed host (that is, a host with more than one IP address), you can use the following method to specify a `stoneHost` argument:

```
System Class >> startReclaimGemForExtentRange: startExtent
to: endExtent onHost: hostNameOrIpAddress stoneHost:
stoneHostNameOrIpAddress
```

This method allows the remote Reclaim GcGem sessions to communicate with the Stone on a separate network connection.

Verify the Reclaim GcGem Configuration

To verify if Reclaim GcGems are running, use one of the following methods:

```
System reclaimGcSessionCount
```

This returns the total number of Reclaim GcGems that are running.

```
System currentGcReclaimSessionsByExtent
```

This returns an array whose size is the number of extents in the repository. The element at each index in the array is the session ID of the Reclaim GcGem for that extent. (A zero indicates that there is no Reclaim GcGem for the corresponding extent.)

To ensure that all extents have Reclaim GcGems:

```
System numberOfExtentsWithoutGC
```

This returns the number of extents that will not have garbage collection performed, because there is no Reclaim GcGem associated with that extent.

```
System numberOfExtentRangesWithoutGC
```

This returns the minimum number of additional Reclaim GcGems that would need to be started in order to have all extents covered by a Reclaim GcGem.

To ensure that all extents have Reclaim GcGems, and that the Admin GcGem is running:

```
System hasMissingGcGems
```

Stopping the Reclaim GcGems

To stop all Reclaim GcGems that are currently running:

```
System stopAllReclaimGcSessions
```

To stop all Reclaim and Admin GcGems that are running:

```
System stopAllGcSessions
```

To configure Reclaim GcGems to not run:

```
System configurationAt: #StnNumGcReclaimSessions put: 0.
```

This also stops all Reclaim GcGems, if they are running.

To Identify Sessions Holding Up Page Reclamation

Privileges required: SessionAccess.

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, reclamation 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
session number: 6   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.

To Tune Reclamation

Parameters to control page reclamation are stored in GcUser's UserProfile. To modify them, log in as GcUser (GcUser's default password is the same as

DataCurator's or SystemUser's default password) 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
%
```

Changes to GcUser's configuration parameters are not soon detected while the Admin GcGem is executing a long-running operation, such as the write set union sweep.

Reclaim Configuration Parameters

The following configuration parameters are available to control the reclaim task, and are stored in GcUser's UserGlobals. For an example of how to change these parameters, see the preceding section.

#reclaimSleepTime	The maximum amount of time in seconds that the process will sleep when no work is scheduled. Must be ≥ 1 ; the default is 10 seconds, maximum 3600.
#sleepTimeBetweenReclaim	The minimum amount of time in seconds that the process will sleep between reclaims, even when work is scheduled. The default is 0 seconds., maximum 3600.
#reclaimMinPages	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 536870911.
#reclaimDeadEnabled	A Boolean indicating whether or not to reclaim dead objects; the default is True.
#objsMovedPerCommitThreshold	The approximate maximum number of live objects to move in a reclaim transaction. Must be ≥ 100 ; the default is 20000, maximum 536870911.
#maxTransactionDuration	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 536870911.

#dataPageBufferSize

The size in pages of the private data page buffer. Must be ≥ 10 ; the default is 64, maximum 1280.

#verboseLogging

Determines if long explicit messages are printed to the GcGem log files. The default is False.

#autoRefreshGcGemConfig

Determines whether the GcGem re-reads its configuration settings from UserGlobals at the end of each transaction. The default is True.

#reclaimDeadShadowPageThreshold

The shadow page threshold at which dead objects are reclaimed. If there are fewer than this number of shadow pages to reclaim, then dead objects are also reclaimed. The default is 1000, maximum 536870911.

#deferReclaimCacheDirtyThreshold

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%.

#enableDebugging

Activates extra consistency checks in the GcGem used for debugging. The default is False.

Cache Statistics

The cache statistics include information about the progress of the reclamation process. The best way to view this data is to start the statmonitor utility and then use VSD (the visual statistics display tool) to view the data. For details about statmonitor and VSD, see Appendix F.

You may find it useful to examine the following statistics:

DeadNotReclaimedKobjs

The number of objects known to be dead but not yet reclaimed. (Units: 1000s of objects.)

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.
PossibleDeadKobjs	The number of objects marked as dereferenced but not yet declared to be dead. (Units: 1000s of objects.)
ReclaimCount	The number of times the page scavenge (reclamation) process has been run.
ReclaimedPagesCount	The number of scavenged pages.

For more information about these and other GemStone statistics, see “Cache Statistics” on page 8-23.

Information is also available by executing the method `System class>>cacheStatistics:` for the Stone’s cache slot.

To Remove References to Large Objects

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.

To Identify Large Objects in the Repository

Ensure that the Topaz display level is sufficient to show the desired information. Use the Topaz `level` command to raise the level to at least 1:

```
topaz 1> level 1
```

The next 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`:

```
topaz 1> printit
AllUsers findObjectsLargerThan: aSize limit: aSmallInt
%
```

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.

If any references to *anObject* reside in Segments for which you do not have read authorization, the last element of the result is the String 'Read Authorization Error encountered'.

If that Array does not provide enough information to track down large repository objects, you can send the same message to System instead:

```
topaz 1> printit
System findObjectsLargerThan: aSize limit: aSmallInt
%
```

NOTE

This method may take considerable time to complete.

This returns an Array of all objects in the repository larger than the SmallInteger *aSize*, whether they are named in a user's symbol list or not. As above, the Array is limited to a maximum of *aSmallInt* elements.

Again, if any references to *anObject* reside in Segments for which you do not have read authorization, the last element of the result is the String 'Read Authorization Error encountered'.

To Search for References to an Object

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*. If an object contains multiple references to *anObject*, that object will appear only once in the resulting Array. The Array is limited to a maximum of *aSmallInt* elements.

The resulting Array contains only those references that reside within Segments for which you have read authorization. If any references to *anObject* reside in Segments for which you do not have read authorization, the last element of the result is the String 'Read Authorization Error encountered'.

You may find this method helpful in locating all instances of a class:

```
topaz 1> printit  
aClassObject findReferencesWithLimit: aSmallInt  
%
```

NOTE

The method findReferencesWithLimit: may take considerable time to complete. In addition, the resulting Array may consume a large amount of disk space.

To limit the disk space required for the result, send the message *anObject findReferences*, which limits the result to a maximum size of 20 elements.

To Remove References to an Object

Complete the process by replacing references to the unneeded object with `nil`. This allows the object to be removed during normal garbage collection. Dereferencing objects must be done through a Smalltalk program.

Appendixes

—
|

GemStone Configuration Options

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 using GemStone Smalltalk to change their internal representation.

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 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 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.

In addition, the GemStone executables **startstone**, **pageaudit**, **topaz**, and **gemsetup** accept command line arguments to point to configuration files:

- The **-z** option sets the system configuration file.
- The **-e** option sets the executable-dependent configuration file.

How GemStone Uses Configuration Files

At start-up 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.

Search for a System-Wide 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.

If GEMSTONE_SYS_CONF *is* defined, GemStone checks to see if it points to a directory.

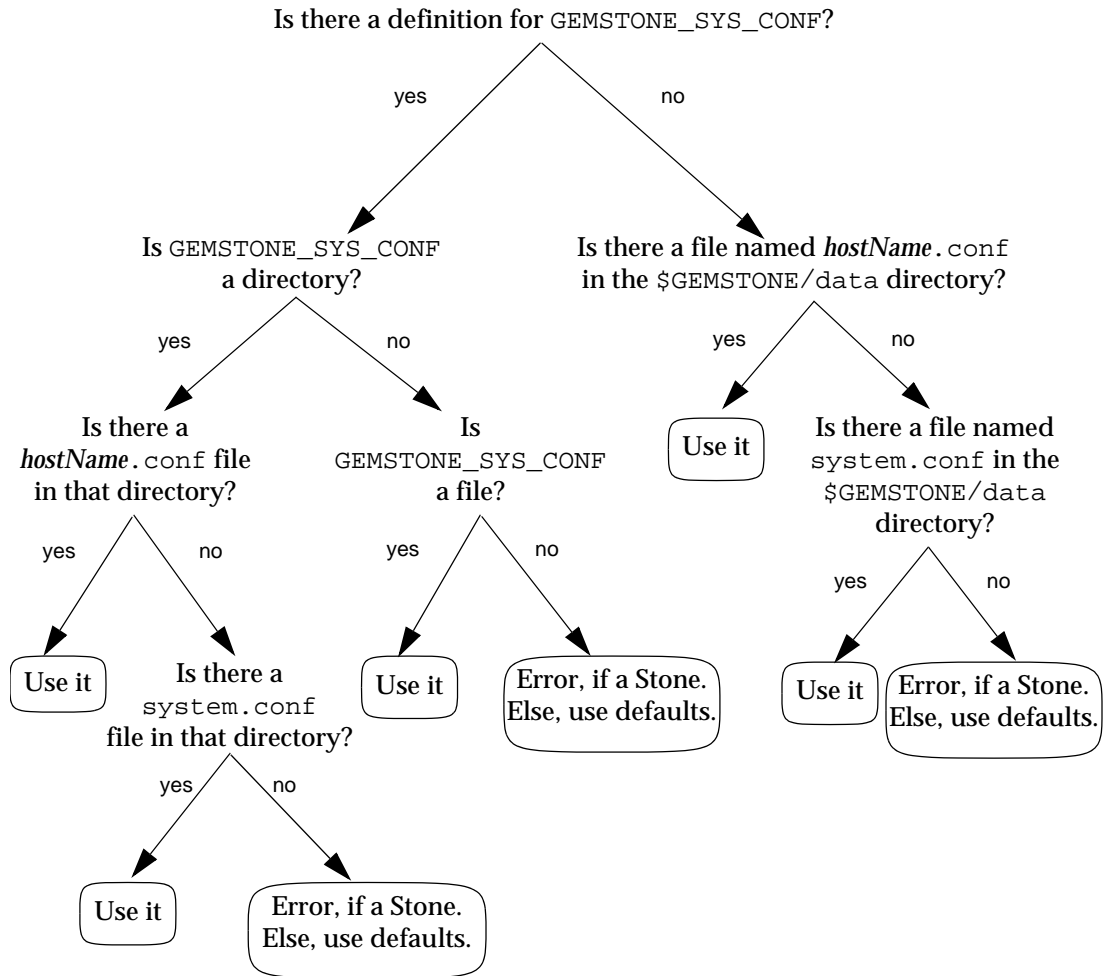
5. If GEMSTONE_SYS_CONF points to a directory, GemStone looks for a file named *hostName.conf* in that directory. If it finds such a file, it uses it. *hostName* must match the results of executing the `hostname` command on the

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-wide and executable-dependent configuration files, the value in the executable-dependent configuration file overrides the system-wide configuration file's value.

Figure A.1 Search Path for a System-Wide Configuration File

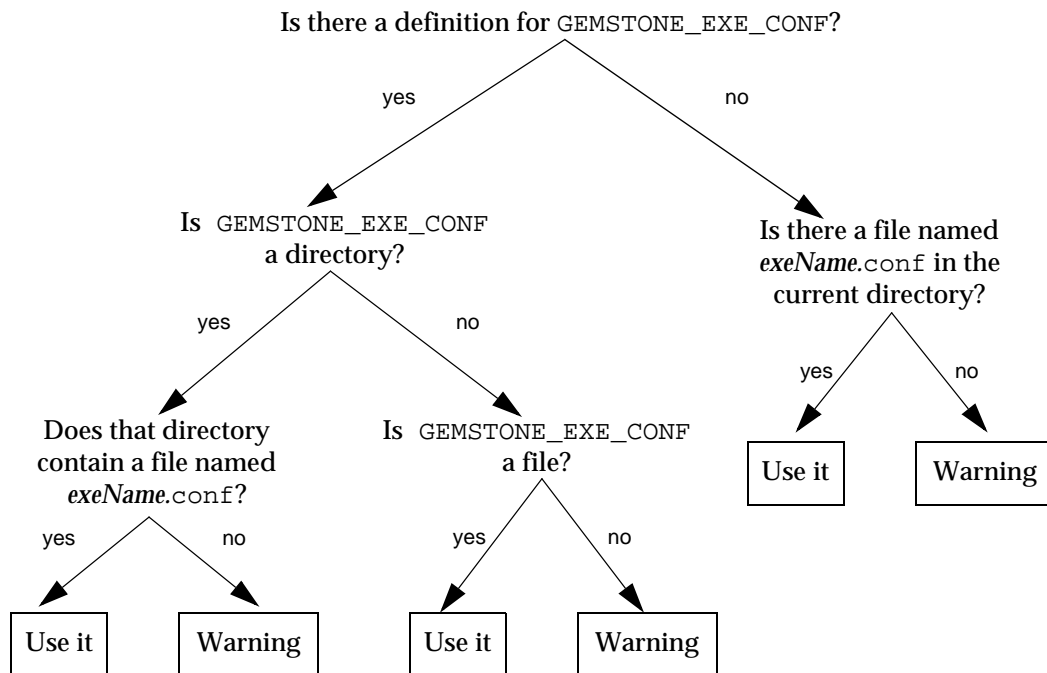


Search for an Executable Configuration File

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 A-7.)
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-wide configuration file for configuration parameters.

Figure A.2 Search Path for an Executable-Dependent 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-wide 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-wide 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-dependent 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-wide 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.

Naming Executable Configuration Files

The default name of an executable configuration file generally is determined from the name of the executable itself.

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 the `gemnetobject` script has been customized. 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.

GcGems

It is sometimes useful to change GcGem parameters in a configuration file specific to the GcGems, so that the changes remain in effect if the system is stopped and restarted. To do so:

Step 1. Copy `$(GEMSTONE)/data/system.conf`.

Step 2. Edit the copy, setting the values you want.

Step 3. Save your changes, renaming the file `gcgem.conf`, `admingcgem.conf`, or `reclaimcgem.conf`, depending on whether the new configuration file is for all GcGems, the Admin GcGem, or Reclaim GcGems. Place the file in GemStone's `bin` directory.

Step 4. Make a copy of the script `$(GEMSTONE)/bin/rungcgem`, and name the copy `$(GEMSTONE)/bin/myruncgem`.

Step 5. Edit `myrungcgem` to specify the customized configuration file. For example, to use a Reclaim GcGem configuration file named `$(GEMSTONE)/data/reclaimgcm.conf`, locate the lines:

```
if [ "$1" = "reclaimgcm" ]; then
    exeConfig="" # path for custom GEMSTONE_EXE_CONF for
    reclaimgc gems
```

and change them as follows:

```
if [ "$1" = "reclaimgcm" ]; then
    exeConfig="$(GEMSTONE)/data/reclaimgcm.conf" # path
    for custom GEMSTONE_EXE_CONF for reclaimgc gems
```

Step 6. Edit the file `$(GEMSTONE)/bin/services.dat`. Comment out the existing line:

```
rungcgem $(GEMSTONE)/sys/rungcgem
```

and add an entry specifying the new script to be executed for the `rungcgem` service.

```
#rungcgem$(GEMSTONE)/sys/rungcgem
rungcgem $(GEMSTONE)/sys/myrungcgem
```

Step 7. Now create the shell script `rungc.sh` to set the environment variable `$(GEMSTONE_EXE_CONF)` to point to the new configuration file `gcm.conf`, `admingcm.conf`, or `reclaimgcm.conf`:

```
$(GEMSTONE_EXE_CONF)=$(GEMSTONE)/bin/gcmname.conf
```

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` so, by default, Gem and Topaz can share the same options. The default location of the file is the user's home directory (`$HOME`).

Linkable GemBuilder Applications

Linkable GemBuilder 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 1-38.

Configuration File Syntax

The following section describes the rules of grammar to be used in editing configuration files.

White space	Leading white space is ignored in the parsing of configuration files. Trailing white space is ignored if it follows the statement termination symbol (;).
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 (#). Comments can be embedded in a configuration file using the following rules: <ul style="list-style-type: none">• by starting a line with the comment symbol• by placing any text after the statement termination symbol (;)

Lists Lists are separated by commas; list elements can be empty, for example:

```
DBF_EXTENT_SIZES = 1999, , 1999;
```

Within lists of values, leading and trailing white space is ignored.

Strings Strings are encased in quotes. An empty string is acceptable in the grammar, and may be expressed by either two double quotes ("") or by no value at all (for instance, OPTION = ;).

Within strings, the escape character is the backslash (\). It can be used as follows:

<u>To generate:</u>	<u>Use the sequence:</u>
backslash (\)	\\
quote (")	\"
statement termination symbol (;)	\;
list separation character (,)	\\,
control characters	\ followed by decimal representation of the character as a zero-padded 3-digit decimal number. For example, the string control-N would read \014, because control-N is ASCII 14.

Case-sensitivity String option values are case-sensitive; boolean option names are *not* case-sensitive.

Maximum Sizes The maximum number of characters allowed for a GemStone configuration option name is 64. 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. For instance,

```
$GEMSTONE/data/extent0.dbf.
```

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.

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. 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. When a warning is issued, the executable ignores the given value and use the option's default value.

Configuration Options

The system-wide 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 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 1-39.

Note that 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, so even if you change the `system.conf` file, you can always recover its original condition.

The following configuration options are listed in alphabetical order.

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 (in MB) 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 null, 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 or until it reaches the maximum size for an extent.

NOTE

The maximum size of an extent is operating system- and platform-dependent, but under no circumstances can be larger than 32 terabytes (32,000,000 MB). For specific information about system dependencies, see the comment in the configuration file for the parameter DBF_EXTENT_SIZES.

For optimal performance using a raw partition, DBF_EXTENT_SIZES should be slightly smaller than the size of the partition so GemStone can avoid having to handle system errors. For example, set it to about 1995 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 file.

Default: EMPTY (no maximum sizes)

Minimum: 1 (MB)

Maximum: operating system- and platform-dependent (see NOTE above)

DBF_PRE_GROW

If DBF_PRE_GROW is set to True, then when a new extent is created, it is grown to its maximum size. If the new extent cannot be grown to the maximum size because of disk capacity problems, then the 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 by the logical repository.

An extent without a maximum size is not pregrown to any size; it is allocated a minimum size determined internally by the GemStone system.

DBF_PRE_GROW, if set to True, will affect existing extents at startup. If an existing extent has a maximum size and that extent is physically not at that maximum size, it is grown to that size and the added portion is initialized. If the grow fails, the extent is reset to its original size and the startup attempt fails.

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.

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.

Units: frames.

Default: -1 (cache size=0 for caches less than 100 MB and 10 for caches of 100 MB or greater)

Minimum: -1

Maximum: 63

GEM_FREE_FRAME_LIMIT

Internal parameter: #GemFreeFrameLimit

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 400 MB or smaller: 10% of the number of frames in the cache
- For primary shared page cache greater than 400 MB: 5000 frames
- For a remote shared page cache: 0

Default: `-1` (see above discussion)

Minimum: 1

Maximum: 65536

GEM_GCI_LOG_ENABLED

This option has no effect in customer executables.

Default: `False`

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 `0` means “never halt”. Ordinarily this option is used only to assist Technical Support in diagnosing problems.

Default: `0`

GEM_IO_LIMIT

Internal parameter: **`#GemIOLimit`**

`GEM_IO_LIMIT` limits the I/O rate to this number of I/Os per second (also applies to linked Gems). Values greater than 1000 result in the I/O rate being limited only by the performance of the underlying file system or disk partitions.

Default: 5000

Minimum: 1

Maximum: 65536

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.

Default: 0
Minimum: 0
Maximum: 10000000

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.

Default: 1000
Minimum: 100
Maximum: 1000000

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.

Units: frames.

Default: -1 (cache size=0 for caches less than 100 MB and 10 for caches of 100 MB or greater)

Minimum: -1
Maximum: 63

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 GEM_FREE_FRAME_LIMIT configuration option (page A-14).

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 400 MB or smaller: 10% of the number of frames in the cache
- For primary shared page cache greater than 400 MB: 5000 frames

To tune the free frame limit of a page server at runtime, use the method `System class>>changeCacheSlotFreeFrameLimit: aSlot to: aValue`.

Default: -1 (see above discussion)
Minimum: -1
Maximum: 65536

GEM_PGSRV_UPDATE_CACHE_ON_READ

Internal parameter: `#GemPgsvrUpdateCacheOnRead`

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 (such Gems do not have a page server).

Default: False

GEM_PRIVATE_PAGE_CACHE_KB

GEM_PRIVATE_PAGE_CACHE_KB sets the size (in KB) of the Gem's private page cache. (This setting also applies to linked Gems.)

Default: 1000
Minimum: 128
Maximum: 524288

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.

Default: 0 (Gem waits forever)

Minimum: 0

GEM_SEND_STN_MSGS_VIA_PGSRV

GEM_SEND_STN_MSGS_VIA_PGSRV specifies whether a remote Gem should send messages to the Stone via the Gem's page server process. Normally, each Gem sends messages to the Stone via a private socket connection. For heavily loaded systems, it is less expensive for the Stone if messages are sent via the Gem's page server.

This option has no effect for Gems running on the same host as the Stone. In that case, communication is performed using shared memory.

Default: True

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 descriptions of the statistics NumSoftRefsCleared, NumLiveSoftRefs, and NumNonNilSoftRefs (page 8-41).

Default: 50

Minimum: 10

Maximum: 80

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 descriptions of the statistics NumRefsStubbedMarkSweep and NumRefsStubbedScavenge (page 8-42).

GEM_TEMPOBJ_CACHE_SIZE

GEM_TEMPOBJ_CACHE_SIZE sets the maximum size (in KB) of the Gem's temporary object memory. (This limit also applies to linked Topaz sessions and linked GemBuilder applications.) This value is set when the VM is initialized and cannot be changed without starting 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.

GEM_TEMPOBJ_CACHE_SIZE defines the maximum memory size. As the actual space required for objects grows, the VM requests and allocates virtual memory as needed.

Default: 10000
Minimum: 2000
Maximum: 1000000

GEM_TEMPOBJ_INITIAL_SIZE

GEM_TEMPOBJ_INITIAL_SIZE specifies the initial amount of memory (in KB) to allocate for temporary object memory, on platforms that do not permit memory to be allocated and not reserved. This parameter affects HP-UX and AIX, but not Solaris or Linux.

During gem execution, when temp obj memory requirements increase, more memory is made available to the gem, up to the limit of GEM_TEMPOBJ_CACHE_SIZE.

When the default setting of -1 is specified, the value of this parameter is computed based on GEM_TEMPOBJ_CACHE_SIZE.

Default: -1
Minimum: 2000
Maximum: GEM_TEMPOBJ_CACHE_SIZE

GEM_TEMPOBJ_MESPACE_SIZE

GEM_TEMPOBJ_MESPACE_SIZE sets the maximum size (in KB) 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.

Default: 0
Minimum: 1000
Maximum: 1000000

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.

Default: 0
Minimum: 16384
Maximum: 524288000

GEM_TEMPOBJ_POMGEN_SIZE

GEM_TEMPOBJ_POMGEN_SIZE sets the maximum size (in KB) 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 the same size as GEM_TEMPOBJ_CACHE_SIZE.

Default: 0
Minimum: 1000
Maximum: 1000000

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 processes that will be started when the shared page cache is created.

Default: 1
Minimum: 0
Maximum: 30

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. Specific operating systems may restrict this action to processes running as root or may require special privileges (such as MLOCK on HP-UX) for this option to take effect; 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_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.

Default: 1900
Minimum: 1
Maximum: 500000

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.

When the default setting of -1 is specified, the value of this parameter is (STN_MAX_SESSIONS + number of extents in repository + 3).

Default: -1

Minimum: 15, or the number extents + 3, whichever is larger

Maximum: determined by STN_MAX_SESSIONS or file descriptor limits

SHR_PAGE_CACHE_SIZE_KB

SHR_PAGE_CACHE_SIZE_KB sets the size (in KB) of the shared page cache. (Additional shared memory is used for overhead.)

Default: 75000

Minimum: 512

Maximum: Limited by system memory and kernel configurations

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

Internal parameter: **#SpinLockCount**

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 4000 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 4000, based upon the number of CPUs detected.

The internal parameter can be changed only by SystemUser.

Default: -1 (use either 1 or 4000, 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 400 MB or smaller: 12.5% of the number of frames in the cache
- For primary shared page cache greater than 400 MB: 7000 frames
- For a remote shared page cache: 1% of the number of frames, or 2000 frames, whichever is smaller

Default: -1 (see above discussion)

Minimum: -1

Maximum: 65536

STN_ADMIN_GC_SESSION_ENABLED

Internal parameter: `#StnAdminGcSessionEnabled`

STN_ADMIN_GC_SESSION_ENABLED determines whether the Admin GcGem is started when the Stone is started. (The Admin GcGem performs administrative garbage collection functions such as write set union sweeps; the Reclaim GcGems perform dead object and page reclamation.)

Default: True

STN_ALLOCATE_HIGH_OOPS

If true, the Stone skips the first 16 million object identifiers and begins to allocate object identifiers (GCI OopTypes) for non-special objects at 16r10000001.

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_CHECKPOINT_INTERVAL

Internal parameter: **#StnCheckpointInterval**

STN_CHECKPOINT_INTERVAL sets the maximum interval between checkpoints. Checkpoints may be written more often, depending on other factors. The unit is seconds.

The internal parameter can be changed only by SystemUser.

Default: 300

Minimum: 5

Maximum: 1800

STN_COMMIT_QUEUE_THRESHOLD

Internal parameter: **#StnCommitQueueThreshold**

STN_COMMIT_QUEUE_THRESHOLD determines whether the Stone defers the disposal of commit records, based on the number of sessions in the commit queue. If the size of the commit queue exceeds this threshold, the Stone defers commit record disposal until the commit queue is less than or equal to the value.

This setting is ignored if the commit record backlog exceeds the value of STN_CR_BACKLOG_THRESHOLD.

Default: -1 (never defer commit record disposal)

Minimum: -1

Maximum: 1024

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. GcGems of all types are exempted from this timeout.

Default: 0 (Stone waits forever)

Minimum: 0

Maximum: 86400

STN_COMMITS_ASYNC

If `STN_COMMITS_ASYNC` is set to `TRUE`, it causes the stone to acknowledge each commit to the requesting session without waiting for the tranlog writes for that commit to complete.

Default: `FALSE`

STN_CR_BACKLOG_THRESHOLD

Internal parameter: `#StnCrBacklogThreshold`

`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 $(2 * \text{STN_MAX_SESSIONS})$. A setting of `0` disables this threshold.

Default: `-1` (Stone waits forever)

Minimum: `-1`

Maximum: `500000`

STN_DISABLE_LOGIN_FAILURE_LIMIT

STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT

Internal parameters: `#StnDisableLoginFailureLimit`,
`#StnDisableLoginFailureTimeLimit`

These options control when a user account is disabled because the user exceeded the `STN_DISABLE_LOGIN_FAILURE_LIMIT` of failed login attempts within the time in minutes specified by `STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT`. 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 internal parameters require the `OtherPassword` privilege.

`STN_LOG_LOGIN_FAILURE_LIMIT`:

Default: `15`

Minimum: `0`

Maximum: `65536`

STN_LOG_LOGIN_FAILURE_TIME_LIMIT:

Default: 15

Minimum: 1

Maximum: 1440 (24 hours)

STN_DISKFULL_TERMINATION_INTERVAL

Internal parameter: **#StnDiskFullTerminationInterval**

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 internal parameter can be changed only by SystemUser.

Default: 3

Minimum: 0 (no sessions are terminated)

Maximum: 1440 (24 hours)

STN_EPOCH_GC_ENABLED

Internal parameter: **#StnEpochGcEnabled**

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.

Default: FALSE

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)

Minimum: 1

Maximum: 1% of the frames in the cache

STN_FREE_SPACE_THRESHOLD

Internal parameter: **#StnFreeSpaceThreshold**

STN_FREE_SPACE_THRESHOLD sets the minimum amount of free space (in MB) to be available in the repository. If the Stone cannot maintain this level by growing an extent, it begins actions to prevent shutdown of the system; for information, see “Repository Full” on page 6-14.

The internal parameter can be changed only by SystemUser.

Default: 1
Minimum: 0 (no threshold)
Maximum: 65536

STN_GEM_ABORT_TIMEOUT

Internal parameter: **#StnGemAbortTimeout**

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 internal parameter can be changed only by SystemUser.

Default: 1
Minimum: 1
Maximum: 1440

STN_GEM_LOSTOT_TIMEOUT

Internal parameter: **#StnGemLostOtTimeout**

STN_GEM_LOSTOT_TIMEOUT sets the time in seconds that the Stone will wait after sending the Gem the error #abortErrLostOtRoot before retracting the Gem’s commit record and forcibly stopping the session. Negative timeouts other than -1 are not allowed. Resolution of timeouts is one half the specified timeout interval.

If the value is -1, the Stone does not stop the Gem; it immediately retracts the session’s commit record, forcing the Gem to completely reinitialize its object caches.

CAUTION

A value of -1 entails a slight risk that the sleeping Gem will reactivate

and begin writing to the shared page cache before it responds to the ABORT_ERR_LOST_OT_ROOT error, thus corrupting the shared page cache.

Because of these risks, design your application to minimize the chances of receiving the ABORT_ERR_LOST_OT_ROOT error.

The internal parameter can be changed only by SystemUser.

Default: 60

Minimum: -1

Maximum: 5000000

STN_GEM_TIMEOUT

Internal parameter: **#StnGemTimeout**

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 internal parameter can be changed only by SystemUser.

Default: 0 (Stone waits forever)

Minimum: 0

STN_HALT_ON_FATAL_ERR

Internal parameter: **#StnHaltOnFatalErr**

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.

The internal parameter can be changed only by SystemUser.

Default: True

Internal settings: 0 = False, 1 = True

STN_LOG_LOGIN_FAILURE_LIMIT

STN_LOG_LOGIN_FAILURE_TIME_LIMIT

Internal parameters: **#StnLogLoginFailureLimit**,
#StnLogLoginFailureTimeLimit

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 internal parameters require the OtherPassword privilege.

STN_LOG_LOGIN_FAILURE_LIMIT:

Default: 10

Minimum: 0

Maximum: 65536

STN_LOG_LOGIN_FAILURE_TIME_LIMIT:

Default: 10

Minimum: 1

Maximum: 1440 (24 hours)

STN_LOOP_NO_WORK_THRESHOLD

Internal parameter: **#StnLoopNoWorkThreshold**

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.

Default: 0
Minimum: 0
Maximum: 536870911

STN_MAX_AIO_RATE

Internal parameter: **#StnMntMaxAioRate**

STN_MAX_AIO_RATE specifies the maximum I/O rate that each AIO page server is allowed when performing asynchronous writes. Since the I/O rate specified is applied to each page server, the total maximum I/O rate on the disk system is this value multiplied by STN_NUM_LOCAL_AIO_SERVERS.

The page servers use this maximum I/O rate for both dirty page and checkpoint writes.

Default: 300
Minimum: 20
Maximum: 2000

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.

The maximum value allowed depends on the maximum allowed by the UNIX kernel. The maximum value for this parameter allowed by GemStone is the value of `_SC_AIO_MAX` or 4096, whichever is lower. On some systems (such as Solaris), it is not possible to determine the value of `_SC_AIO_MAX`. In that case, GemStone will impose a maximum value of 128. Otherwise the maximum is 4096 or `_SC_AIO_MAX`, whichever is lower.

For further information on the `_SC_AIO_MAX` kernel parameter, refer to the UNIX documentation for your system and/or the UNIX man page for the `sysconf()` call.

Default: 128
Minimum: 100
Maximum: the minimum of 4096 or `_SC_AIO_MAX`; or 128 (see above)

STN_MAX_SESSIONS

STN_MAX_SESSIONS limits the number of simultaneous sessions (number of Gem logins to Stone), excluding SymbolGem and GcGems. 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. This parameter is provided so the number of users to be restricted to avoid overloading the host computer. The maximum number of file descriptors per process (imposed by the operating system kernel) can also limit the maximum number of sessions.

If you increase STN_MAX_SESSIONS, and you are not allowing the system to calculate SHR_PAGE_CACHE_NUM_PROCS, you will need to increase that parameters as well.

Recommended: 40 or more, depending on your requirements

Default: 40

Minimum: 1

Maximum: 10000

STN_MAX_VOTING_SESSIONS

Internal Parameter: **#StnMaxVotingSessions**

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.

Default: 100

Minimum: 1

Maximum: 1000000

STN_NUM_GC_RECLAIM_SESSIONS

Internal parameter: **#StnNumGcReclaimSessions**

STN_NUM_GC_RECLAIM_SESSIONS specifies the number of page reclaim garbage collector gems (Reclaim GcGems) that will be started when the Stone starts. The maximum number of Reclaim GcGems is equal to the number of extents as specified in DBF_EXTENT_NAMES. If you specify a value larger than the number of extents, one GC reclaim will be started for each extent.

Default: 1
Minimum: 0
Maximum: 256

STN_NUM_LOCAL_AIO_SERVERS

STN_NUM_LOCAL_AIO_SERVERS is the approximate number of page server processes to start as local asynchronous I/O servers for the shared page cache on the node where the Stone runs. The number of extents known to the Stone at startup is divided by the value of STN_NUM_LOCAL_AIO_SERVERS to compute the internal configuration parameter `StnRdbfMaxFilesPerServer`. The latter parameter is the approximate number of extent files to be serviced by each AIO page server.

For instance, if your configuration has six extents, setting STN_NUM_LOCAL_AIO_SERVERS to 3 causes each AIO page server to service $(6 \div 3) = 2$ extents.

Under certain circumstances, multiple AIO page servers can help you achieve the maximum possible commit rate. A value greater than one is recommended only if there are two or more extents, the host has multiple CPUs (to allow parallel execution), and the disk drive hardware allows concurrent writes to disk (the extents are on separate spindles, or the equivalent).

Default: 1
Minimum: 1
Maximum: 256

STN_OBJ_LOCK_TIMEOUT

Internal Parameter: **#StnObjLockTimeout**

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 the methods `System >> waitForRcWriteLock:` and `System >> waitForApplicationWriteLock:queue:autoRelease:`.

Default: 0 (stone waits forever)
Min: 0
Max: 86400

STN_PAGE_REMOVAL_THRESHOLD

Internal Parameter: **#StnPageRemovalThreshold**

`STN_PAGE_REMOVAL_THRESHOLD` 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. The stone cache statistic `PagesNeedRemovingThreshold` reflects the current value of this parameter.

Default: 40
Minimum: 0
Maximum: 1792

STN_PRIVATE_PAGE_CACHE_KB

`STN_PRIVATE_PAGE_CACHE_KB` sets the default size (in KB) of the Stone page cache.

Default: 2000
Minimum: 128
Maximum: 524288

STN_REMOTE_CACHE_TIMEOUT

Internal parameter: `#StnRemoteCacheTimeout`

`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 internal parameter can be changed only by `SystemUser`.

Default: 5
Minimum: 0
Maximum: 5000000

STN_SHR_TARGET_PERCENT_DIRTY

Internal parameter: `#StnShrPcTargetPercentDirty`

`STN_SHR_TARGET_PERCENT_DIRTY` specifies the maximum percentage of the Stone's shared page cache that can contain dirty pages without AIO page servers increasing their IO rates.

Default: 20
Minimum: 1
Maximum: 90

STN_SIGNAL_ABORT_CR_BACKLOG

Internal parameter: `#StnSignalAbortCrBacklog`

STN_SIGNAL_ABORT_CR_BACKLOG sets the number of old transactions (commit records) above which the Stone will start to generate SignalAbort messages to a Gem that is running outside of a transaction. You may need to tune this option according to your application's commit rate and your system's tolerance for the possible swapping activity caused by awakening sleeping session processes.

The internal parameter can be changed only by SystemUser.

Default: 20
Minimum: 0
Maximum: 65536

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. 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. 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. See also STN_TRAN_LOG_DEBUG_LEVEL, which can cause old log files to be retained under partial logging.

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 7, “Managing Transaction Logs.”

Default: None. The system will not run unless you provide a value.
Initial setting: False

STN_TRAN_LOG_DEBUG_LEVEL

This option is only for GemStone internal use. Customers should not change the default setting. Values ≥ 2 inhibit removal of old transaction logs when in partial logging mode.

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.

Default: Empty (the system will not run without at least two entries)

Minimum: 2 entries

Maximum: 2^{31} entries

Initial setting: \$GEMSTONE/data/, \$GEMSTONE/data/

STN_TRAN_LOG_LIMIT

Internal parameter: #StnTranLogLimit

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.

The internal parameter can be changed only by SystemUser.

Default: 1000

Minimum: 25

Maximum: 1000

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 (in MB) of all 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. The sizes also apply to the corresponding entries in STN_REPL_TRAN_LOG_DIRECTORIES when that list is not empty.

For transaction logs on raw partitions, if the size specified is larger than the physical size of the corresponding raw partition, the STN_TRAN_LOG_SIZES value is adjusted appropriately.

Default: Empty (the system will not run unless sizes are specified)

Minimum: 3

Maximum: 16384

Initial setting: 100, 100

STN_TRAN_Q_TO_RUN_Q_THRESHOLD

Internal parameter: `#StnTranQToRunQThreshold`

STN_TRAN_Q_TO_RUN_Q_THRESHOLD specifies the number of sessions in the commit queue (waiting for the commit token) above which the stone will allow the remaining sessions in the queue to process unions (read old commit records) while waiting for the commit token.

For example, if this parameter is set to 6 and there are 9 sessions in the commit queue, the 3 last sessions will be allowed to process unions while waiting for the token. If there are 6 or fewer sessions in the queue, no sessions will process unions.

The first session in the commit queue never processes unions since it will receive the token when the current commit completes.

Default: 6

Minimum: 1

Maximum: 1024

Miscellaneous Internal Parameters

The internal parameters described in this section can be read by using the method `System class>>configurationAt:`.

The parameter `#StnLoginsSuspended` requires the `SystemControl` privilege. All other parameters can be changed only by `SystemUser`.

#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.

#SessionInBackup

`#SessionInBackup` is the GemStone session number of the session performing a full backup, or -1 if a backup is not in progress.

#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`.

#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.

#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.

#StnMaxSessions

`#StnMaxSessions` is the maximum allowed number of sessions for your GemStone license, as set by the GemStone keyfile. The value of this parameter is 0 when not limited by the keyfile. This internal parameter is not related to the `STN_MAX_SESSIONS` configuration option.

#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.

#StnTranLogOriginTime

`#StnTranLogOriginTime` is the time when the current transaction log was started.

—
|

GemStone Utility Commands

The GemStone/S 64 Bit utility commands in this appendix are provided in the `$GEMSTONE/bin` directory. For most of these commands, you can display usage information by entering the command name with the `-h` option. For example:

```
% copydbf -h
```

```
Usages: copydbf srcNRS dstNRS [-h|-l|-m|-P] [-C] [-E] [-f filePrefix]
        [-n netldi] [-p pgsvrId] [-s Mbytes]
...
copydbf srcNRS [-i|-I] [-n netldi] [-p pgsvrId]
...
```

UNIX man pages are available for more detailed information on each command.

copydbf

copydbf	<i>sourceNRS destinationNRS</i> [-C] [-E] [-filePrefix] [-nnetLdiName] [-ppgsvrId] [-sMbytes] [-h -l -m -P]
copydbf	<i>sourceNRS</i> [-i -I] [-nnetLdiName] [-ppgsvrId]
<i>sourceNRS</i>	The source file or raw partition (containing an extent, a transaction log, or a full backup) as a GemStone network resource string. Use of a tape device as the source is supported only for a GemStone full backup.
<i>destinationNRS</i>	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 <i>destinationNRS</i> 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. Use of a tape device as the destination is not supported.
-C	Compress output. The output must be a filesystem file. Write the output compressed, in gzip format. The output file name will have the suffix .gz appended to it, if it does not end in .gz.
-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.
-filePrefix	If <i>destinationNRS</i> is a file system directory, then <i>filePrefix</i> overrides the filename prefix that would be generated based on the contents of <i>sourceNRS</i> . If <i>destinationNRS</i> is other than a file system directory, this option has no effect.
-nnetLdiName	The name of the GemStone network server; the default is gs64ldi.

-ppgsvrId	The name of a specific runpgsvr (similar to gemnetid).
-sMB	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.
-h	Displays a usage line (as shown on page B-1) and exits.
-i	Information only. When this option is present without <i>destinationNRS</i> , information about <i>sourceNRS</i> is printed without performing a file copy. If both -i and <i>destinationNRS</i> 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.
-l	Least-significant-byte ordering for the <i>destinationNRS</i> . This byte ordering is the native byte ordering for Intel processors.
-m	Most-significant-byte ordering for the <i>destinationNRS</i> . This byte ordering is the native byte ordering for Solaris SPARC, AIX POWER, and HP-UX PA-RISC processors.
-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.

The **copydbf** utility requires exclusive access to repository files in order to copy them without corruption. You must shut down the Stone repository monitor before copying an extent. You may copy any transaction log that is not the active log.

GemStone repository files on the UNIX file system can usually be copied using the ordinary **cp** command. You can use the first form of **copydbf** for disk-to-disk copies between machines (without NFS) or copies to and from raw disk partitions. You must give an NRS (network resource string) for both the source file and the destination. (A local machine filespec is a subset of an NRS.)

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 standard error before starting the copy. The size of the destination file is printed to standard error after the copy is completed. For example:

```
% copydbf $GEMSTONE/data/extent0.dbf .
```

```
Source file: /users/GemStone/data/extent0.dbf
  file type: extent  fileId: 0
  byteOrder: Sparc (MSB first) compatibilityLevel: 840
  Last checkpoint written at: 03/31/07 16:20:29 PDT.
Destination file: ./extent0.dbf
  byteOrder: Sparc (MSB first)
  Clean shutdown, no tranlog needed for recovery,
  last tranlog written to had fileId 5 ( tranlog5.dbf ).
  File size is 23.1 MBytes (1408 records).
```

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: Sparc (MSB first) compatibilityLevel: 830
  Last checkpoint written at: 03/31/07 15:30:30 PDT.
  Oldest tranlog needed for recovery is fileId 5
  ( tranlog5.dbf ).
```

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
```

```
Source file: backup.dat
  file type: backup  fileId: 0
  The previous file last recordId is -1.
Destination file: /dev/null
  Full backup started from checkpoint at: 03/31/07 15:28:37
PDT.
  Oldest tranlog needed for restore is fileId 5
  ( tranlog5.dbf ).
```

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: Sparc (MSB first) compatibilityLevel: 830
  The file was created at:      03/31/07 15:28:18 PDT.
  The previous file last recordId is 36 .
  Scanning file to find last checkpoint...
Destination file: /dev/null
  byteOrder: Sparc (MSB first)
  Checkpoint 1 started at: 03/31/07 15:28:18 PDT.
    oldest transaction references fileId -1 ( this file ).
  Checkpoint 2 started at: 03/31/07 15:28:37 PDT.
    oldest transaction references fileId -1 ( this file ).
  Checkpoint 3 started at: 03/31/07 15:30:30 PDT.
    oldest transaction references fileId -1 ( this file ).
  Checkpoint 4 started at: 03/31/07 15:57:01 PDT.
    oldest transaction references fileId -1 ( this file ).
  File size is 35328 bytes (69 records).
```

To pre-allocate disk space in the destination file, use the **-s** option. For instance, **-s10** would allocate at least 10 MB to the created file. The output file is made as short as possible by default.

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 \  
!tcp@node#auth:username@password#dbf!/users/extent0.dbf_copy
```

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	[-h -l -p -x] [-c] [-q] [-v] [-t secs] [-u user] [-m host]+ [[-n] name]+
-h	Prints a usage line and exits.
-l	Prints a long listing (includes pid and port).
-p	Prints only the pid (process id), or 0 if the server does not exist.
-x	Prints an exhaustive listing, with each item on a separate line.
-c	Removes locks left by servers that have been killed.
-q	(Quiet.) Don't print any extra information; intended for use when the output will be processed by some other program.
-v	Verify the status of each server.
-t secs	Wait <i>secs</i> seconds for server to respond (only with -v); default is 2 seconds.
-u user	Only list servers started by <i>user</i> .
-m host	Only list servers on machine <i>host</i> ; default is '.' which represents the local host.
[-n] name	Only list the server <i>name</i> .

The **gslist** command prints information about GemStone servers. The default listing prints the following server attributes in columns:

Status	One of the following: OK server is accepting clients (-v only) frozen server is not responding (-v only) full server can't accept any more clients (-v only) exists server process exists but is not verified killed server process does not exist
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, or cache (shared page cache monitor)

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 page A-2).

execonf The GemStone executable configuration file (see page A-5).

GEMSTONE Root of the product tree used by the server.

To specify multiple hosts, include the **-m host** option for each host. To specify the local host explicitly, you can use "**-m .**".

If the name of a server is printed and the server is not on the local machine, the name is prefixed by `host :`, where *host* is the name of the remote machine. For **gslist** to list servers on a remote host, **gslist** must (in most cases) be able to contact a NetLDI running on the remote machine. If both the local host and the remote host are running Windows, then a remote NetLDI is not needed.

To specify multiple server names, include the **-n name** option for each server. Because the **-n** switch is optional, just a name on the command line also specifies a server name.

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.

If many servers are reported as **frozen**, try increasing `-t secs`.

pageaudit

pageaudit	[<i>gemStoneName</i>] [- <i>eexeConfig</i>] [- <i>zsystemConfig</i>] [- f] [- h]
<i>gemStoneName</i>	Name of the GemStone repository monitor; the default is <code>gs64stone</code> . Network resource syntax is not permitted.
- <i>eexeConfig</i>	The GemStone executable-dependent configuration file (see page A-5).
- <i>zsystemConfig</i>	The GemStone system configuration file (see page A-2).
- f	Keeps running beyond the first error, if possible.
- h	Displays a usage line and exits.

Audit the pages in a GemStone repository, which must not be in use. **pageaudit** opens the repository specified by the relevant configuration files. The three arguments *gemStoneName*, *-eexeConfig*, and *-zsystemConfig* determine which configuration files **pageaudit** reads. (For more information, see “How GemStone Uses Configuration Files” on page A-2.)

The options for this command generally are not needed for a standard GemStone configuration.

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 “To Perform a Page Audit” on page 8-10.

removedbf

removedbf	<i>dbfNRS</i> [- n <i>netLdiName</i>] [- p <i>pgsvrNetId</i>] [- h]
<i>dbfNRS</i>	The GemStone repository filename or device, as a network resource string, for the repository to be removed.
- n <i>netLdiName</i>	The name of the GemStone network server; default is <i>gs641di</i> .
- p <i>pgsvrNetId</i>	The name of a specific page server to use.
- h	Displays a usage line and exits.

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 and on remote machines. You must provide the NRS (network resource string) of the file to be removed. (A local machine filespec is a subset of an NRS.)

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.

This command does not disconnect an extent from the logical repository. To alter the configuration by disconnecting an existing extent, see “How to Remove an Extent” on page 6-7.

The options for this command generally are not needed for a standard GemStone configuration.

startcachewarmer

startcachewarmer	[-d -D] [-h] [-l<i>limit</i>] [-n<i>numGems</i>] [-p<i>password</i>] [-s<i>stone</i>] [-u<i>userID</i>] [-w<i>delayTime</i>] [-W]
-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.
-h	Displays a usage line and exits.
-l<i>limit</i>	Stops cache warming if the number of free frames in the cache falls below the specified <i>limit</i> . If <i>limit</i> is -1 (the default), have the system compute the actual limit based on the size of the shared cache. If <i>limit</i> is 0, force cache warming to continue even if the shared cache is full.
-n<i>numGems</i>	Number of Gem sessions to start (default: 1).
-p<i>password</i>	GemStone password for logging in Gems (default: 'swordfish').
-s<i>stone</i>	Name of the running Stone (default: <code>gs64stone</code>).
-u<i>userID</i>	GemStone user for logging in Gems (default: 'DataCurator').
-w<i>delayTime</i>	Wait <i>delayTime</i> seconds between spawning Gems (default: 1)
-W	Wait for cache warming Gems to exit before exiting this script. By default, this script spawns Gems in the background and exits immediately.

The **startcachewarmer** command warms up the shared page cache on startup, by preloading object and 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.

Cache warming runs in one or two phases:

- In the first phase, the object table is loaded into the shared page cache. During this phase, if any data pages will be loaded later, the data pages that are referenced by the object table lookups are recorded for use in phase 2.

- In the second phase, data pages remembered from the first phase are loaded either into the shared page cache or the file buffer. This phase runs only if data pages will be loaded into the shared page cache or the file buffer.

—
|

startnetldi

startnetldi	<code>[netLdiName] [-llogFile] [-ttimeout] [-aname] [-p<i>low:high</i>] [-n] [-g -s] [-d] [-h]</code>
<i>netLdiName</i>	The name of the GemStone network server. The name may contain digits but it must not be entirely numeric. Network resource syntax is not permitted. The default is <code>gs64ldi</code> .
-llogFile	The logged output of the NetLDI; the default is <code>/opt/gemstone/log/NetLdiName.log</code>
-ttimeout	Seconds to wait for a spawned client to start, such as a Gem; default is 120 seconds.
-aname	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.
-p<i>low:high</i>	The low and high ports in the pool of ports to use in creating processes.
-n	Do not create any <i>ad hoc</i> processes (ad hoc processes are ones not listed in <code>\$GEMSTONE/bin/services.dat</code>).
-g	Guest mode; no accesses are authenticated. This option is not allowed if the NetLDI's effective user id is the root account.
-s	Secure; require authentication for all NetLDI accesses.
-d	Debug mode; inserts more extensive messages in the log file.
-h	Displays a usage line and exits.

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. On slow or heavily loaded machines, use a larger timeout value, typically 300 seconds.

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 `$/GEMSTONE/sys/netldid` does NOT have the S bit set. Log in as the captive account *name*, then issue this command:

```
% startnetldi -g -aname
```

For information about authentication modes, see “How to Arrange Network Security” on page 3-9.

For assistance with startup failures, refer to “To Troubleshoot NetLDI Startup Failures” on page 4-9.

startstone

startstone	<i>[gemStoneName]</i> [- <i>llogFile</i>] [- <i>eexeConfig</i>] [- <i>zsystemConfig</i>] [- h] [- R] [- N] [- C] [- s]
<i>gemStoneName</i>	Name of the GemStone repository monitor, default is <code>gs64stone</code> . Network resource syntax is not permitted.
-llogFile	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/<i>gemStoneName</i>.log</code>
-eexeConfig	The GemStone executable-dependent configuration file (see page A-5).
-zsystemConfig	The GemStone system configuration file (see page A-2).
-h	Displays a usage line and exits.
-R	Start up from the most recent checkpoint and go into restore-from-transaction-logs state. This option should be used only for recovery using operating system backups.
-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.
-C	Convert extents from GemStone/S 2G to GemStone/S 64 Bit. This option is provided only for use during repository upgrades, as described in the <i>GemStone/S 64 Bit Release Notes</i> .
-s	Linux only; disable use of the POSIX AIO library, <code>libposix-aio.so</code> , and use <code>librt.so</code> instead. Use of this option will result in slower performance.

This command opens the GemStone repository specified by the relevant configuration files. The three arguments *gemStoneName*, **-eexeConfig**, and **-zsystemConfig** determine which configuration files **startstone** reads. (For more information, see “How GemStone Uses Configuration Files” on page A-2.)

The options for this command generally are not needed for a standard GemStone configuration.

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. In such case, **-N** forces the repository to start up anyway, even though transactions committed since the last checkpoint will be lost. If the Stone detects that the logs actually are present, it performs a normal startup. If a transaction log file is present but corrupted, you may have to remove that log file before restarting.

You can use the **-N** option to start a Stone if the transaction files have been lost or corrupted, but the extents are still available. A new transaction log will be initialized as part of the startup.

The **-R** option is intended for use when the repository is restored by starting GemStone on an operating system backup of the extents. The repository monitor is left in a state equivalent to that following restoration of a GemStone backup. You must then invoke Topaz to restore from transaction logs (if available) 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 assistance with startup failures, refer to “To Troubleshoot Stone Startup Failures” on page 4-3.

stopnetldi

stopnetldi *[netLdiName]* [-h]

netLdiName The name of the GemStone network server; the default is gs641di. Network resource syntax is not permitted.

-h Displays a usage line and exits.

Gracefully stop a GemStone network server. The argument to this command is optional, and are not needed for a standard GemStone configuration.

stopstone

stopstone	<i>[gemStoneName [gemStoneUserName [gemStonePassword]]]</i> [-i] [-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> .
-i	Causes any current GemStone sessions to be terminated immediately.
-h	Displays a usage line and exits.

Gracefully shut down a GemStone repository monitor. In the process, a checkpoint is performed in which all committed transactions are written to the extents and to any replicates. If you specify the **-i** (immediate) option, the repository monitor is shut down even if there are GemStone users logged in. 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 4-14.

topaz

topaz [-r]	[-i] [-nnetLdiName] [-h]
topaz -l	[-i] [-nnetLdiName] [-exeConfig] [-zsystemConfig] [-TtocSizeKB] [-h]
-l	Invoke the linked version of Topaz.
-r	Invoke the RPC (remote procedure call) version of Topaz.
-i	Ignore the initialization file, <code>.topazini</code> .
-n netLdiName	The name of the GemStone network server; the default is (1) the setting of the <code>GEMSTONE_NRS_ALL</code> environment variable; (2) <code>gs64ldi</code>
-exeConfig	The GemStone executable-dependent configuration file (applies only to linked sessions). See page A-5.
-zsystemConfig	The GemStone system configuration file (applies only to linked sessions). See page A-2.
-TtocSizeKB	The <code>GEM_TEMPOBJ_CACHE_SIZE</code> that will be used. Overrides any settings provided in configuration files passed as arguments with the <code>-e</code> or <code>-z</code> options. Only applies to linked sessions.
-h	Displays a usage line and exits

This command invokes various forms of Topaz. The default is to invoke the remote procedure call (RPC) version of Topaz. The arguments `-exeConfig` and `-zsystemConfig` determine the configuration files that **topaz -l** reads. For more information about this, see “How GemStone Uses Configuration Files” on page A-2.

The arguments to this command are optional, and are not needed for a standard invocation of Topaz. For further information, see the *GemStone Topaz Programming Environment*.

waitstone

waitstone	<code>[gemStoneName netLdiName] [timeout]</code>
<i>gemStoneName</i>	The name of the GemStone repository monitor, commonly <code>gs64stone</code> .
<i>netLdiName</i>	The name of the GemStone network server, commonly <code>gs64ldi</code> .
<i>timeout</i>	How many minutes 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.

This command reports whether the GemStone repository *gemStoneName* is ready to accept logins or whether *netLdiName* is ready to accept requests. During the first 10 seconds, **waitstone** tests every two seconds to see if the Stone or NetLDI is ready; thereafter, a new connection is attempted once every 10 seconds. When the service is ready, **waitstone** issues a message to `stdout`. If the service is not ready by the time the specified number of minutes have elapsed, **waitstone** reports an error.

You may specify the *gemStoneName* and *netLdiName* arguments as a GemStone network resource string. (See Appendix C, "Network Resource String Syntax.")

This command returns 0 exit status if the operation is successful; otherwise, it returns a non-zero value.

waitstone does not have a `help` argument, but you can type:

```
man waitstone
```

to display the online manual page.

Network Resource String Syntax

This appendix describes the syntax for network resource strings. A network resource string (NRS) provides a means for uniquely identifying a GemStone file or process by specifying its location on the network, its type, and authorization information. GemStone utilities use network resource strings to request services from a NetLDI.

Overview

One common application of NRS strings is the specification of login parameters for a remote process (RPC) GemStone application. An RPC login typically requires you to specify a GemStone repository monitor and a Gem service on a remote server, using NRS strings that include the remote server's hostname. For example, to log in from Topaz to a Stone process called "gs64stone" running on node "handel", you would specify two NRS strings:

```
topaz> set gemstone !@handel!gs64stone
topaz> set gemnetid !@handel!gemnetobject
```

Many GemStone processes use network resource strings, so the strings show up in places where command arguments are recorded, such as the GemStone log file. Looking at log messages will show you the way an NRS works. For example:

```
Opening transaction log file for read,
filename = !tcp@oboe#dbf!/user1/gemstone/data/tranlog0.dbf
```

An NRS can contain spaces and special characters. On heterogeneous network systems, you need to keep in mind that the various UNIX shells have their own rules for interpreting these characters. If you have a problem getting a command to work with an NRS as part of the command line, check the syntax of the NRS recorded in the log file. It may be that the shell didn't expand the string as you expected.

NOTE

Before you begin using network resource strings, make sure you understand the behavior of the software that will process the command.

See each operating system's documentation for a full discussion of its own rules about escaping certain characters in NRS strings that are entered at a command prompt.

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 !tcp@oboe#auth:user@password!gs64stone
% waitstone "!tcp@oboe#auth user@password!gs64stone"
```

Defaults

The following items uniquely identify a network resource:

- Communications protocol— such as TCP/IP
- Destination node—the host that has the resource
- Authentication of the user—such as a system authorization code
- Resource type—such as server, database extent, or task
- Environment—such as a NetLDI, a directory, or the name of a log file
- Resource name—the name of the specific resource being requested.

A network resource string can include some or all of this information. In most cases, you need not fill in all of the fields in a network resource string. The

information required depends upon the nature of the utility being executed and the task to be accomplished. Most GemStone utilities provide some context-sensitive defaults. For example, the Topaz interface prefixes the name of a Stone process with the **#server** resource identifier.

When a utility needs a value for which it does not have a built-in default, it relies on the system-wide defaults described in the syntax productions in “Syntax” on page C-4. You can supply your own default values for NRS modifiers by defining an environment variable named GEMSTONE_NRS_ALL in the form of the *nrs-header* production described in the Syntax section. If GEMSTONE_NRS_ALL defines a value for the desired field, that value is used in place of the system default. (There can be no meaningful default value for “resource name.”)

A GemStone utility picks up the value of GEMSTONE_NRS_ALL as it is defined when the utility is started. Subsequent changes to the environment variable are not reflected in the behavior of an already-running utility.

When a client utility submits a request to a NetLDI, the utility uses its own defaults and those gleaned from its environment to build the NRS. After the NRS is submitted to it, the NetLDI then applies additional defaults if needed. Values submitted by the client utility take precedence over those provided by the NetLDI.

Notation

Terminal symbols are printed in boldface. They appear in a network resource string as written:

#server

Nonterminal symbols are printed in italics. They are defined in terms of terminal symbols and other nonterminal symbols:

username ::= nrs-identifier

Items enclosed in square brackets are optional. When they appear, they can appear only one time:

address-modifier ::= [protocol] [@ node]

Items enclosed in curly braces are also optional. When they appear, they can appear more than once:

nrs-header ::= ! [address-modifier] {keyword-modifier} !

Parentheses and vertical bars denote multiple options. Any single item on the list can be chosen:

```
protocol ::= ( tcp | serial | default )
```

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 ::= [protocol] [ @ node ]
```

Specifies where the network resource is.

```
protocol ::= ( tcp | serial | default )
```

Supports heterogeneous connections by predicating address on a network type. If no protocol is specified, `GCI_NET_DEFAULT_PROTOCOL` is used. On UNIX hosts, this default is **tcp**.

```
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:

```
!tcp@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 user on the target network. A valid password is needed only if the resource type requires authentication. #encrypted is used by GemStone utilities. If no authentication information is specified, the system will try to get it from the .netrc file. This type of authorization is the default.

username ::= *nrs-identifier*

If no user name is specified, the default is the current user.
(See the earlier discussion of *nrs-identifier*.)

password ::= *nrs-identifier*

If no password is specified, the system will try to obtain it from the user's .netrc file. (See the earlier discussion of *nrs-identifier*.)

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* (page C-4) 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* on page C-4.)

#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" (defined below) is used. (See the definition of *nrs-identifier* on page C-4.)

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
%M machine's network node name
%N executable's base name
%P process pid
%U user name
%% %

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/bin/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.

Users

The AllUsers object, an instance of UserProfileSet, contains the UserProfile objects of the users that are known to the repository. Initially, it has the following elements: **SystemUser**, **DataCurator**, **GcUser**, **SymbolUser**, and **Nameless**. *You must never delete these users.*

The **SystemUser** UserProfile is ordinarily used only for performing GemStone system upgrades. Certain system objects — including the GemStone-supplied kernel classes, along with their methods and class variables — are owned by the SystemUser and are stored in the System Segment. (That Segment also contains all instances of classes Character and SmallInteger.)

CAUTION

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 operations except those that **require** SystemUser privileges, such as upgrade and full restores.*

The **DataCurator** UserProfile is used to perform the data curator tasks described in the this manual. Most of the predefined GemStone objects listed in this section are owned by the DataCurator and are stored in the DataCurator Segment.

The **GcUser** UserProfile is used to control GemStone's garbage collection tasks. A person does not normally log into GemStone as **GcUser**. **GcUser** initially has only the GarbageCollection privilege. Its UserGlobals dictionary contains the parameters that control garbage collection.

The **SymbolUser** UserProfile is used to run a background SymbolGem process that creates all new Symbols, based on session requests that are managed by the Stone. A person does not normally log into GemStone as **SymbolUser**. SymbolUser initially has only the GarbageCollection privilege. Its UserGlobals dictionary contains the collection AllSymbols.

The **Nameless** UserProfile is for use only by other GemStone products. Do not use this account or change it unless instructed to do so by GemStone.

Dictionaries

UserGlobals. Each UserProfile object contains a symbol list, an Array of SymbolDictionaries that initialize a session so that it can resolve symbols at compile-time. The first element in the symbol list is always the SymbolDictionary UserGlobals, which initially contains two keys: #UserGlobals (corresponding to the dictionary itself) and #NativeLanguage (initially English). Each user's UserGlobals dictionary is stored in that user's default Segment.

Globals. This SymbolDictionary defines the GemStone kernel classes and any other objects to which all GemStone users may need to refer. Table D.1 (on page D-6) lists the contents of this dictionary. For information about the kernel classes, see the class comments in the image. The Globals dictionary is stored in the DataCurator Segment.

Published. This SymbolDictionary can be used to share objects among users. The Published dictionary is primarily an example, so it provides minimal object deployment capability. The Published dictionary is stored in the Published Segment.

Non-Numeric Constants

The following non-numeric constants are defined in the Globals dictionary and stored in the System Segment:

- **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 stored in the System Segment. Refer to IEEE standards 754-1987 and 854-1987 for more information regarding their meanings in floating-point calculations.

DecimalPlusInfinity
DecimalMinusInfinity
DecimalPlusQuietNaN
DecimalMinusQuietNaN
DecimalPlusSignalingNaN
DecimalMinusSignalingNaN
PlusInfinity
MinusInfinity
PlusQuietNaN
MinusQuietNaN
PlusSignalingNaN
MinusSignalingNaN

Repository and Segments

SystemRepository. This Repository is defined in the Globals dictionary. The SystemRepository object itself is stored in the DataCurator Segment, and its indexable part contains references to all GemStone Segments.

GemStone represents all the disk space it uses as a single instance of class Repository. When GemStone is first installed, that Repository has the name SystemRepository. (The GemStone Smalltalk message `name:` can be used to subsequently change the name of the system Repository.) The SystemRepository object initially contains eight segments, three of which are

public and named: the SystemSegment (owned by the SystemUser), the DataCuratorSegment (owned by the DataCurator), and the PublishedSegment (owned by SystemUser). New Segments may be created (added to the SystemRepository object) when new users are added.

SystemSegment. This Segment is defined in the Globals dictionary, and is referenced from the indexable part of the SystemRepository. The SystemSegment object itself is stored in the DataCurator Segment.

The SystemSegment is the default Segment for its owner, the SystemUser (who has write authorization for any of the objects in this Segment). The “world” (that is, the set of all GemStone users) is authorized to read, but not write, the objects in this Segment. In addition, the group #System is authorized to write in this Segment.

DataCuratorSegment. This Segment is defined in the Globals dictionary, and is referenced from the indexable part of the SystemRepository. The DataCuratorSegment object itself is stored in the DataCurator Segment.

The DataCuratorSegment is the default Segment for its owner, the DataCurator (who has write authorization for any of the objects in this Segment). The “world” (that is, the set of all GemStone users) is authorized to read, but not write, the objects in this Segment. In addition, the #DataCuratorGroup is are authorized to write in this Segment.

Objects in the DataCuratorSegment include the Globals dictionary, the SystemRepository object, most Segment objects, AllUsers (the set of all GemStone UserProfiles), AllGroups (the collection of groups authorized to read and write objects in GemStone segments), and each UserProfile object.

PublishedSegment. This Segment is defined in the Globals dictionary, and is referenced from the indexable part of the SystemRepository. The PublishedSegment object itself is stored in the DataCurator Segment.

The PublishedSegment is owned by the SystemUser. The group #Subscribers is authorized to read in this Segment. The group #Publishers is authorized to read and write in this segment. The “world” is not authorized to read or write the objects in this Segment.

DbfHistory. DbfHistory is a String object that contains information regarding conversions and updates applied to the repository.

NativeLanguage. This Symbol is defined in the Globals dictionary (with an initial value of #English), and may be redefined in each user’s UserGlobals directory. The NativeLanguage object permits GemStone to return error messages and

other interactive messages in any of the supported languages. (Initially, only English is supported.)

Global Collections

AllGroups. This CanonicalStringDictionary is defined in the Globals dictionary, and is stored in the DataCurator Segment. Each Symbol in AllGroups corresponds to a group of users. When GemStone is first installed, AllGroups contains the symbols #System, #Publishers, #Subscribers, and #DataCuratorGroup.

AllUsers. The AllUsers object (a UserProfileSet) is defined in the Globals dictionary, and is stored in the DataCurator Segment. AllUsers contains the UserProfiles of all GemStone users. When GemStone is first installed, AllUsers contains five UserProfiles: SystemUser, DataCurator, GcUser, SymbolUser, and Nameless.

AllClusterBuckets. This ClusterBucketArray is defined in the Globals dictionary, and is stored in the DataCurator Segment. Each Symbol in AllClusterBuckets corresponds to a cluster bucket, which organizes a physical storage specification for a group of objects. When GemStone is first installed, AllClusterBuckets contains symbols for 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. Making this bucket invariant increases the fault tolerance of the system, and facilitates both building the kernel and repository conversion.
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 SymbolKeyValueDictionary is defined in the Globals dictionary, and is stored in the System Segment. 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.

ErrorSymbols. This SymbolDictionary is defined in the Globals dictionary, and is stored in the System Segment. It maps mnemonic symbols to error numbers.

GemStoneError. This SymbolDictionary is defined in the Globals dictionary, and is stored in the DataCurator Segment. 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.

InstancesDisallowed. This IdentitySet is defined in the Globals dictionary, and is stored in the System Segment. It is a collection of the GemStone classes for which you cannot create instances. Some of these classes (like System) have no instances at all, for others all possible instance already exist (like Boolean).

Table D.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	Boolean
	#nil	UndefinedObject
	#true	Boolean
Repository and Segments	#DataCuratorSegment	Segment
	#DbfHistory	String
	#MinutesFromGMT	SmallInteger
	#NativeLanguage	Symbol
	#PublishedSegment	Segment
	#SystemRepository	Repository
	#SystemSegment	Segment

Table D.1 Initial Contents of the Globals Dictionary (Continued)

	Key	The object's class
Collections	#AllClusterBuckets	ClusterBucketArray
	#AllGroups	CanonicalStringDictionary
	#AllUsers	UserProfileSet
	#ConfigurationParameterDict	SymbolKeyValueDictionary
	#ErrorSymbols	SymbolDictionary
	#GemStoneError	SymbolDictionary
	#Globals	SymbolDictionary
	#InstancesDisallowed	IdentitySet
GemStone Internal Objects	#ANSIException	SymbolDictionary
	#AsciiCollatingTable	ByteArray
	#ConversionReservedOopMap	Array
	#ConversionStatus	Array
	#DoubleByteAsciiCollatingTable	DoubleByteString
	#GcCandidates	RcQueue
	#GcCandidatesCount	RcCounter
	#GcHints	UndefinedObject
	#GcWeakReferences	Array
	#GemStoneRCLock	Object
	#GsIndexingSegment	Segment
	#ImageVersion	SymbolDictionary
	#OldAllUsers	AbstractUserProfileSet
	#RcTreeNode	UndefinedObject
	#_remoteNil	UndefinedObject
	#SecurityDataSegment	Segment
	#SharedDependencyLists	DepListTable
#VersionParameterDict	SymbolKeyValueDictionary	
plus all kernel classes		

Current TimeZone

Each instance of `Date`Time 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 must be installed as part of application installation or configuration. 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'
```

- **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. To get updated source files, download from

```
ftp://elsie.nci.nih.gov/pub/tzdata*.tar.gz
```

Shipped files are based on tzdata2006p.tar.gz. The timezone sources in this file may be compiled using the zic timezone compiler, which GemStone provides as a convenience (see "zic" on page 10).

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 that these are not GemStone utilities, and support for these is not provided by GemStone.

You may download the source code for these utilities here:

```
ftp://elsie.nci.nih.gov/pub/tzcode*.tar.gz
```

Shipped files are based on tzcode2006p.tar.gz. 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 you to interactively select a time zone. The interactive script asks you a series of questions about the current location and outputs the resulting time zone description to standard output. 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.

Specifying an invalid zone name to `zdump` does NOT return an error; instead, it returns the `zdump` output for GMT. This reflects the same behavior of the time routines in `libc`.

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 yearistype] [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.

To create files in a specific location, rather than the standard platform directory (on Solaris, `/usr/share/lib/zoneinfo`), use the `-d` *directory* option.

For example, to recompile sources on Solaris to the GemStone timezone database, execute the following:

```
zic -d $GEMSTONE/pub/timezone/etc/zoneinfo/
  /usr/share/lib/zoneinfo/src/northamerica
```

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 GcGem logs. By default, Admin GcGem 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 GcGem log files are created in a different directory.

GEMSTONE_CHILD_LOG

Used by parent process to tell child process the name of its log file. To keep the child's log from being deleted when the process terminates normally, unset

this variable in the appropriate script, such as
`$GEMSTONE/sys/gemnetobject`.

GEMSTONE_EXE_CONF

The location of an executable-dependent configuration file; see “Creating an Executable Configuration File” on page A-6.

GEMSTONE_LOG

The location of system log files for the Stone repository monitor and its child processes. For further information, see “GemStone System Logs” on page 8-2.

GEMSTONE_MAX_FD

Limits the number of file descriptors requested by a GemStone process. For further information, see “Estimating File Descriptor Needs” on page 1-13.

GEMSTONE_NRS_ALL

Sets a number of network-related defaults, including the type of user authentication that GemStone expects. For further information, see “To Set a Default NRS” on page 3-15.

GEMSTONE_PAGE_MGR_LOG_DIR

The directory location for Page Manager Gem logs. By default, Page Manager 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 Page Manager Gem log files are created in a different directory.

GEMSTONE_RECLAIM_GC_LOG_DIR

The directory location for all Reclaim GcGem logs. By default, Reclaim GcGem 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 GcGem log files are created in a different directory.

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 A-2.

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. If you are facing this situation, GemStone/S Technical Support will recommend a new value for this variable and work with you to analyze the problem.

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_KEEPALIVE

A non-empty string disables the network keepalive facility. For further information about keepalive, see "Disrupted Communications" on page 3-7.

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_WRITE_CORE_FILE

By default, core files are not written 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.

`_POSIX_OPTION_ORDER`

`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`

`netldinn`

`runpgsvr`

—
|

statmonitor and VSD Reference

This appendix provides details about VSD and statmonitor:

- How to use the statmonitor and Visual Statistics Display (VSD) utilities to evaluate GemStone performance.
- What options are available on the statmonitor command line.
- A VSD menu reference.
- Descriptions of VSD files, including the syntax for template filters.

Using statmonitor and VSD

Every commercial aircraft in service today contains a flight data recorder, commonly called a black box. Black boxes are used to help determine what went wrong if the aircraft ever crashes. Without it, crash investigators would have very little to go on and the cause of many crashes would never be known.

In a production GemStone application, the *statmonitor* program serves the role of the black box. It runs in the background, logging the values of all cache statistics to a text file. Should a problem with the repository occur, you can review these statistics to determine the cause, allowing you to identify problems and tune GemStone applications for better performance.

Many questions about repository performance are impossible to answer after the fact unless statmonitor log files are available. For example:

- Why did garbage collection take longer than normal?
- Why did the backup run slower than before?
- Why was there more repository growth than normal?
- Which Gem session was consuming the most CPU or I/O during a time of poor performance?
- Which Gem had the most number of commit failures?

If you're trying to answer questions such as these, statmonitor statistics files are your best source of information. statmonitor reads the statistics from the shared page cache that are recorded by GemStone processes, and writes those statistics to a file.

To help you interpret the statistical information that statmonitor gathers, GemStone provides a tool called VSD (Visual Statistical Display). VSD's graphical user interface gives you a meaningful way to see how your GemStone system changes over time, based on periodic samples of the system's state. VSD reads the statmonitor files and displays the values in a graph. VSD can also start statmonitor, if it is not already running, and read the values as soon as statmonitor collects them.

statmonitor and VSD are companion tools:

- *statmonitor* is a standalone executable with a command-line interface. It does not log into GemStone nor use a GemStone session. However, it is version-sensitive; you must use the version of statmonitor appropriate for your GemStone version.
- VSD provides a graphical user interface for viewing the text files that statmonitor produces. It is version-independent: any version of VSD can read any flat file produced by any version of statmonitor.

On UNIX platforms, VSD requires X-Windows.

The statmonitor tool uses a discrete sampling algorithm; events that occur between samples are not recorded.

Starting VSD and statmonitor

To start statmonitor, at the command line, enter:

```
statmonitor stoneName -f outputFile -i sampleInterval
```

For complete statmonitor command line syntax, see page F-14.

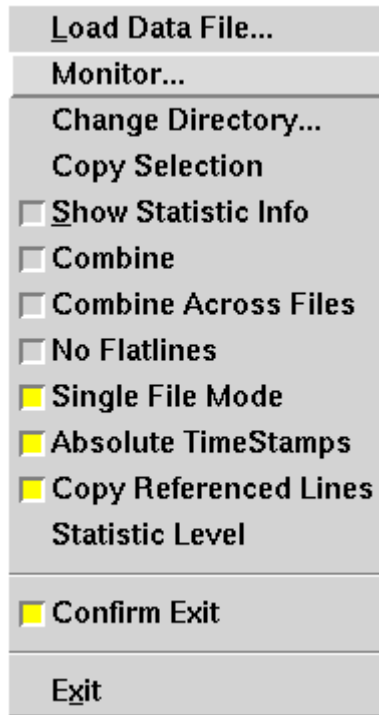
To start VSD, simply execute the `vsd` command. (This assumes that `$GEMSTONE/bin` is defined in your path.)

The initial startup screen is similar to that shown in Figure F.1 on page F-4.

Loading an existing statmonitor output file

To load a statmonitor file into VSD, do one of the following:

- To browse for an existing statmonitor output file, choose **Load Data File** from the **Main** menu.



NOTE: Online help is available for each of these menu items; see page F-5.


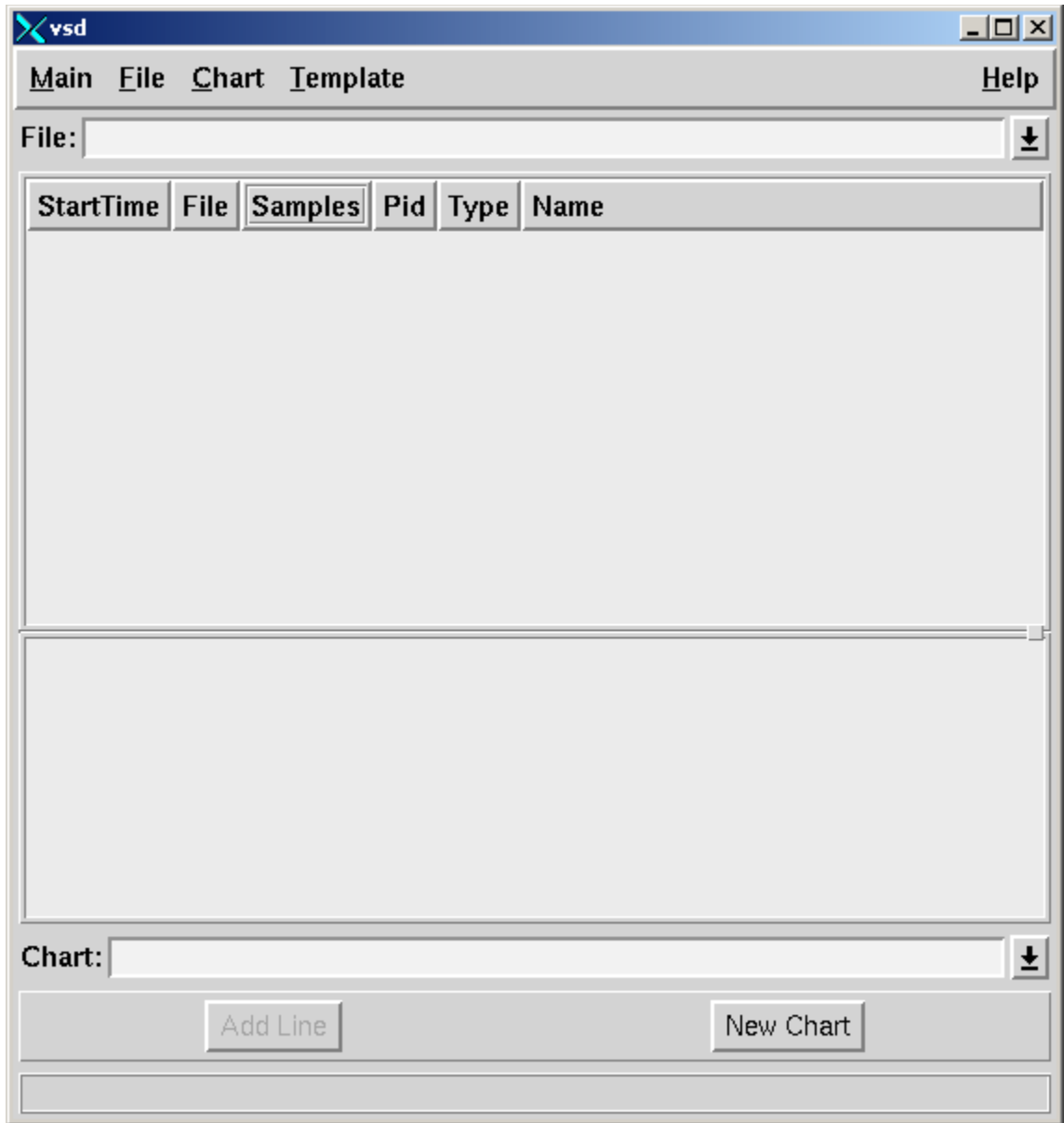
- If you know the name of the load file, you can type the full path in the File entry box (Figure F.1), then press Enter.
- To switch to a statmonitor output file that you've already loaded, click the down-arrow next to the File entry box, then select a file from the list. 

Figure F.1 VSD Startup Screen



Viewing VSD online help

VSD provides extensive online help about the individual menu options, along with guidance on how to perform various tasks. Use the **Help** menu (see Figure F.1 on page F-4) to display help for menu options in the Main window or in the Chart window (discussed later in this chapter).

Maintaining a current view of the data file

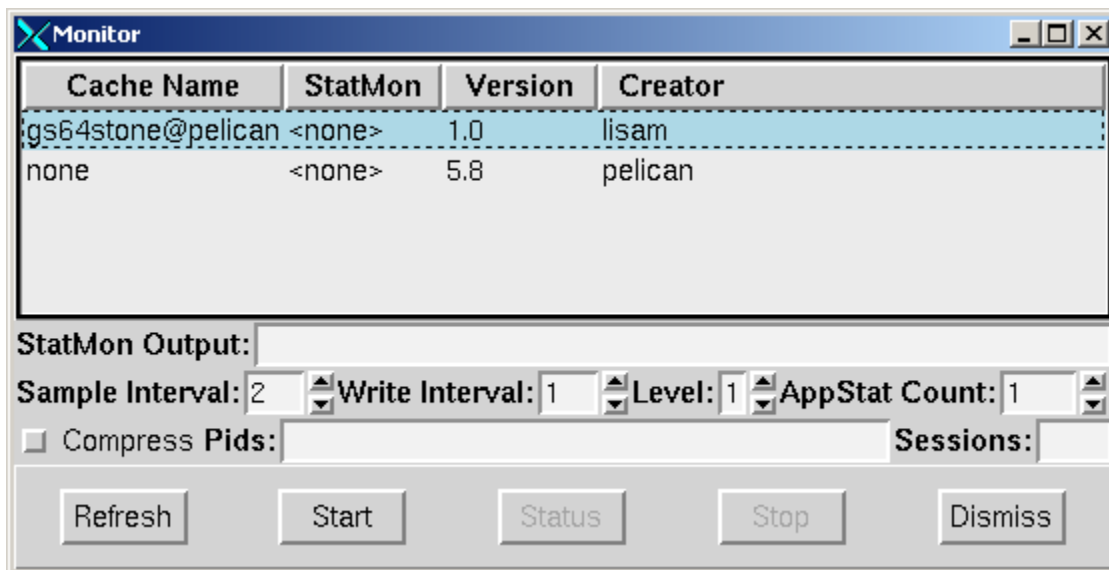
If you select **File > Auto Update**, VSD automatically updates your display, and any associated charts, any time the data file changes. Otherwise, you can choose **File > Update** periodically to update the displays when you choose.

Starting statmonitor from VSD and viewing current data

You can also use VSD to start statmonitor and read directly from the data file that statmonitor is currently creating. To do so:

Step 1. Choose **Monitor** from the **Main** menu (page F-3).

Step 2. When the Monitor window appears, click to select the cache on the local machine for which you want to obtain statistics.



Step 3. If necessary, modify any statmonitor startup parameters:

- The name of the statmonitor output file.
- The sample interval (in seconds)—that is, how frequently to read the cache.
- The write interval—the maximum number of seconds to wait before flushing the cached information to the output file.
- The level of operating system statistics to gather.
- The number of application statistics to gather.
- Whether to compress the output file.
- The GemStone sessionId of each session that you want to monitor. If you don't specify the sessionId explicitly, all sessions will be monitored.
- The type of sessions that you want to monitor.

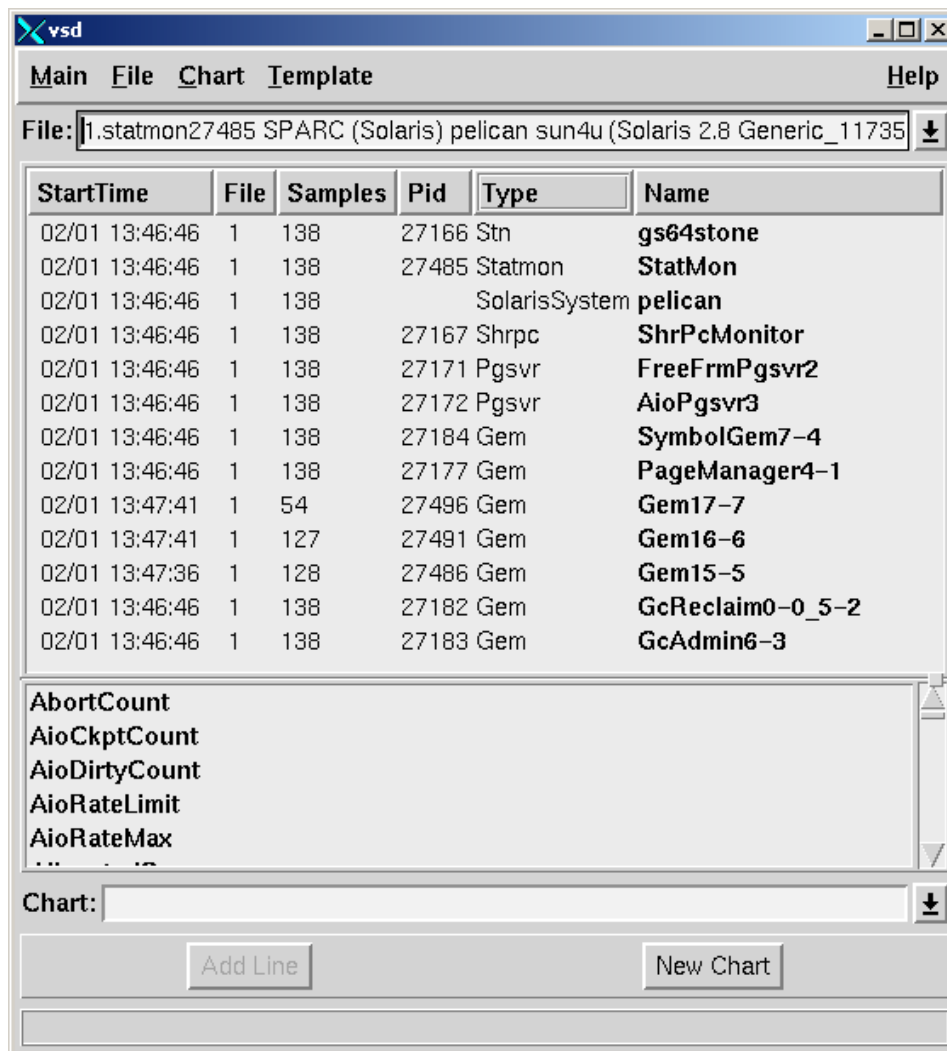
Step 4. When you're done, click **Start**.

VSD loads the data file as soon as it has some samples in it (unless you have explicitly turned off Auto Update, as described on page F-5).

Step 5. When you're finished collecting statistics, click **Stop**.

Viewing Statistics

After you've loaded the data file, the VSD window looks something like this:



Your next step is to select the specific cache statistics to view. Statmonitor keeps statistics for various GemStone processes, corresponding to the cache slots described on page 8-23. They appear as follows under the heading **Type** in the process list:

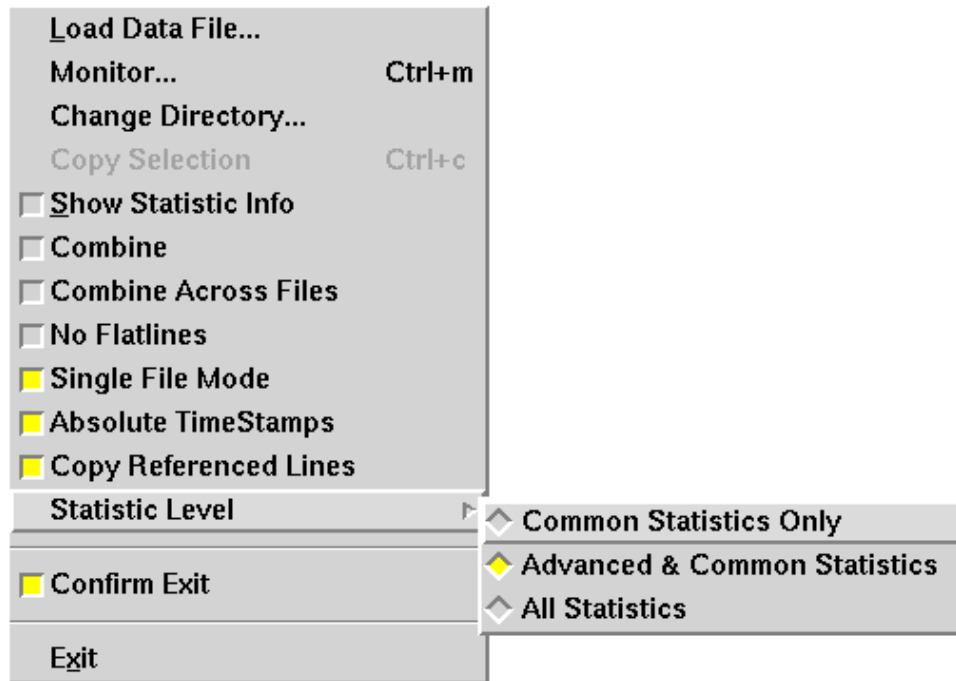
Stn	Stone
Gem	Gem
Shrpc	shared page cache monitor
Pgsvr	AIO page server
FLPgsvr	free frame page server
Statmon	statmonitor
Appstat	application-defined statistics

NOTE

For detailed descriptions of each statistic, see “Cache Statistics”, beginning on page 8-23.

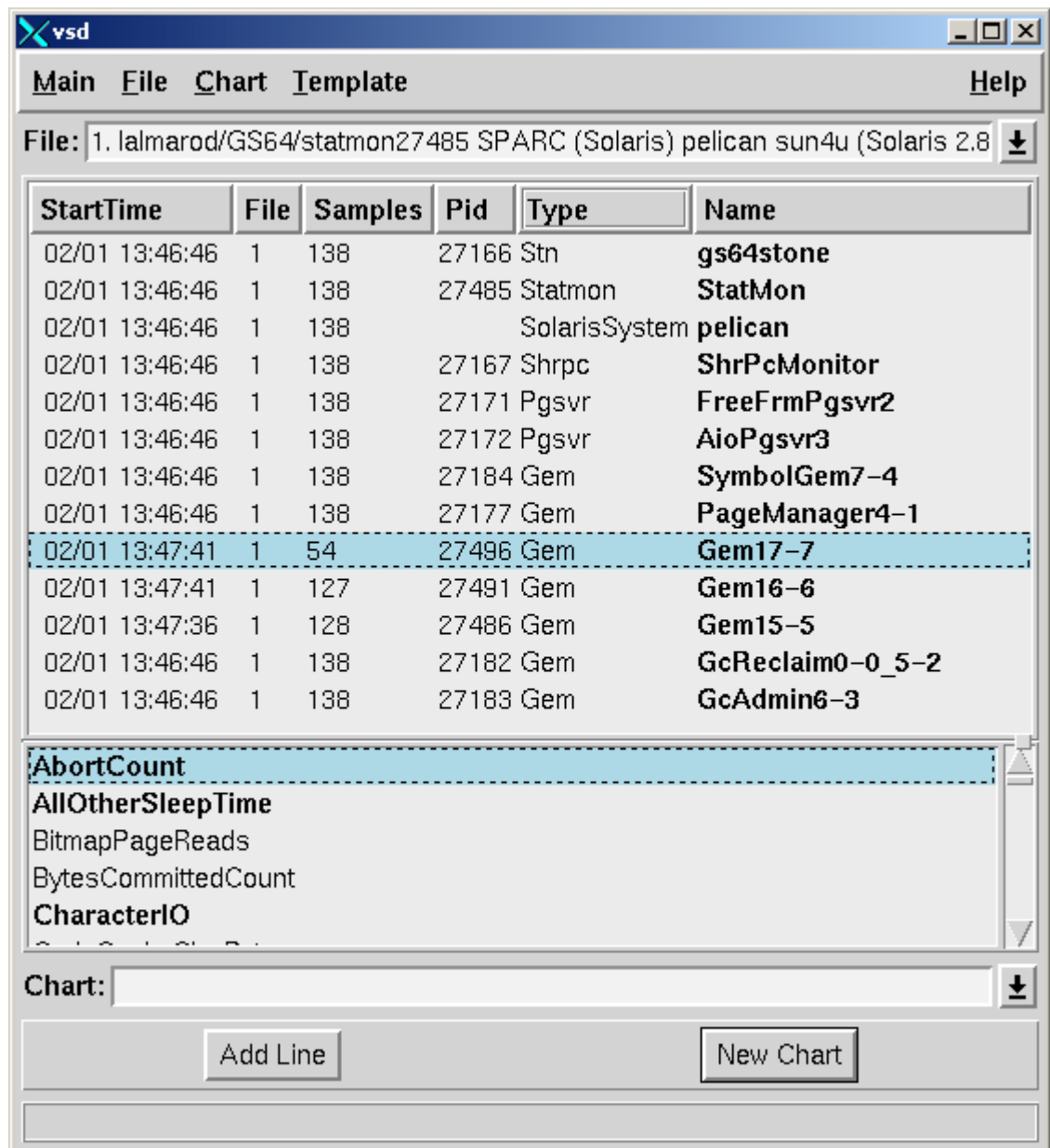
Step 1. Each statistic, or *counter*, has a characteristic called a *level* reflecting the amount of background knowledge about GemStone processes needed to use it with understanding. You can set up VSD to list (in its main window and in associated charts) only those statistics that are at, or below, a certain level of complexity.

To establish the levels of statistics that you want to display in VSD, choose **Statistic Level** from the VSD window's **Main** menu:



Step 2. Now, you're ready to select one or more processes. To do so, in the process list (Figure F.2 on page F-10), click the left mouse button on the process(es) you wish to view.

Figure F.2 Process List with Selected Gem



Instead of selecting a single process in this way, you can use the right mouse button menu in the VSD process list to perform these useful functions:

- Search for a specific session name or process ID.

To find a specific process, click the mouse in the process list, then press Ctrl-S. When the dialog box appears, enter the PID or name of the process you're looking for. VSD highlights the first process with that PID or name. To find the next match, enter Ctrl-S again.

To select the highlighted item, click on it. When you're done, press Return.

- Select all processes, all Gem processes, or all page server processes.
- Combine multiple processes into a single chart line.

This can be quite helpful. For example, if you want to measure page-reads per second for several hundred Gem processes, you can select all Gem processes, then combine them into a single line in the chart, thus rendering the data much more readable.

- Eliminate *flatlines*—processes whose values are always zero.
- Work with multiple statmonitor files simultaneously.
- Select whether to use absolute timestamps in the display (useful when merging files) or relative timestamps (useful when comparing files).

Step 3. Next, select a statistic from the list just below the process list.

Step 4. After you've selected the statistic, do one of the following:

- To display the statistic in an existing chart, click the down-arrow and select the chart name in the **Chart** entry box. Then click on **Add Line**.
- To display the statistic in a new chart, type the name of the new chart in the **Chart** entry box, then click **New Chart**.

NOTE

If you don't explicitly specify a chart name, VSD assigns one for you.

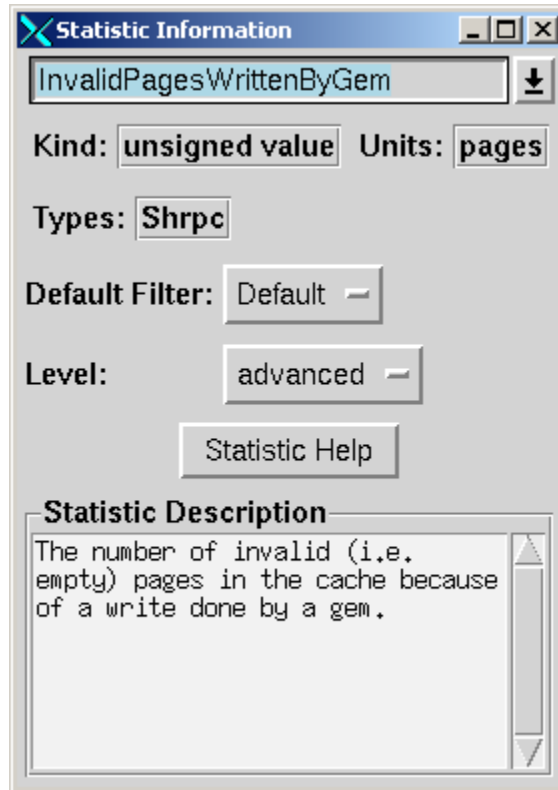
Step 5. To add another statistic to the chart, repeat steps 3 and 4.

Customizing Your Chart

You can customize and manipulate a VSD chart in many ways:

- To select a line in the chart, click on it or on its entry in the chart legend.

- To delete a line from the chart, click the middle mouse button (if available) on its entry in the chart legend. Or select the line and choose **Line > Delete**.
- To find out about a specific point in a chart, hold the mouse pointer over it.
- To view an explanation about the most recently selected statistic, go to the VSD main window and choose **Show Statistic Info** from the **Main** menu. You'll see a Statistic Information window that looks something like this:



In the Statistic Information window, you can redefine the level and default filter for any VSD statistic.

TIP

If you leave this window up as you work, it changes to reflect the current statistic. In this way, you can get a quick explanation of any statistic you're currently examining.

-
-
- To aid in identifying processes, you can customize the name used for a specific Gem or Topaz session. To do so:

Step 1. Log into the Gem or Topaz session as soon after you've started it as possible.

Step 2. At the Topaz prompt or in a GemStone workspace, evaluate this expression:

```
System _cacheName: 'myName'
```

Filter

Whenever you add a line to a chart, the filter determines how much information is displayed for the selected statistic:

- **Default** — *None* or *PerSecond*, as appropriate for the statistic.
- **None** — Displays the values for the statistic as they are expressed in the statmonitor text file.
- **PerSample** — Displays the difference between two consecutive samples of the statistic.
- **PerSecond** — Displays the difference between two consecutive samples of the statistic, divided by the number of elapsed seconds between the two samples.
- **Aggregate** — Displays a running total for the statistic. Each raw value from previous samples is added to the current one.

Once you've added the line to a chart, you can override its default filter by specifying a new filter from the drop-down menu at the top of the Chart window.

Using VSD Chart Templates

VSD templates let you quickly add a set of lines to a chart. Templates are helpful if you find yourself performing the same task frequently in VSD — for example, monitoring the same five or six statistics. By creating a template for the statistics that you want to monitor most frequently, you can automate the task of building charts.

In your template, you can assign a filter for each statistic, to determine how much information is displayed for that statistic. You can also restrict the template to look for extreme conditions (for example, Gem processes that are consuming 90% or more of the CPU).

VSD is shipped with a set of predefined templates, maintained in the `.vsdtemplates` file in your home directory.

NOTE

You can use a text editor to change or delete templates in the `.vsdtemplates` file. For details about the format of the `.vsdtemplates` file, see “VSD Files” on page F-16.

To do this...

Create a new chart from a template.

Apply a template to the chart that you’re viewing.

Reread the template file `.vsdtemplates` into VSD after you’ve edited it.

Save the current chart as a template.

do this...

In the VSD main window, choose **New Template Chart** from the **Template** menu.

This is a good way to display some of the more useful system statistics.

In the Chart window, choose **Add From Template** from the **Chart** menu.

NOTE

If you have zoomed in on a chart, the template filter is applied only to values within the zoomed range.

In the VSD main window, choose **Reload Template File** from the **Template** menu.

In the Chart window, configure the chart as you desire, then choose **Save Template** from the **Chart** menu. When you exit VSD, the template will be saved to the `.vsdtemplates` file.

If you save the current chart as a template, you may still need to edit the `.vsdtemplates` file so that you can get the selected Gem or page server.

Statmonitor command line syntax

statmonitor *stonename* **-f** *fileName* [*options*]

-f *fileName*

The output filename. By default, the output filename is `statmonN.out`, where *N* is the process ID. To send output to stdout instead of a file, specify **-f stdout**.

-z

Write the output in compressed gzip format.

-
-
- | | |
|-------------------------|--|
| -r | Restart a new output file when the current one completes. Each file is given a unique name. The -h and -t arguments determine when a restart is done. |
| -i interval | The interval in seconds (default is 20). Select either -i or -I . |
| -I intervalMs | The interval in milliseconds (default is 20000, minimum is 100). Select either -i or -I . |
| -u seconds | The maximum number of seconds to wait before flushing the cached information to the output file (default is 60). If you specify -u 0 , then the flush will be done every interval. |
| -h hours | The maximum number of hours to write to the output file before starting a new one (default is forever). Select either -h or -t . |
| -t times | The maximum number of samples to collect before starting a new output file (default is forever). Select either -h or -t . |
| -m stoneHostName | The stone host name (default is localhost). |
| -n numAppStats | The number of application statistics (default is 0). |
| -p sessionId | A GemStone sessionId to monitor. Can be repeated. Default: monitor all sessions. |
| -s level | <p>The level of system statistics to collect. The default <i>level</i> is 1, which limits system statistics collection to GemStone processes. (Also collect statistics for the operating system.)</p> <p>When <i>level</i> is 2, also collect statistics for physical disks and TCP.</p> <p>When <i>level</i> is 3, also collect statistics for disk partitions and network interfaces.</p> <p>To disable the collection of system statistics, set <i>level</i> to 0.</p> |
| -S | Sample only the Stone and shared cache monitor. |
| -P | Sample the Stone, shared cache monitor, and all AIO page servers only. |

VSD Files

VSD writes the following files to your home directory. This information is useful primarily to users who want to do more complex configuration or work directly with template files.

.vsdrc

This file is used by VSD to record its configuration. VSD reads the file when it starts and writes the file when it exits. You should not modify this file manually. If you want to specify configuration values that will be retained from one session to the next, do so in `.vsdconfig` instead.

.vsdconfig

You can configure VSD by setting default values in this file. Any value set in `.vsdconfig` will override the same definition in `.vsdrc`.

If you delete the `.vsdconfig` file, then the next time VSD is started, the file will be recreated with default values. If a value in this file contains whitespace, you must enclose it in braces. For example:

```
set vsd(exec:rm) {rm -f}
```

.vsdtemplates

This file contains VSD chart templates. A template is a list of line specs that define the content and appearance of a VSD chart. Each line spec has the following format:

```
{type name stat scale filter axis [statFilter]}
```

type — One of the following: `Stn`, `Shrpc`, `Gem`, or `Pgsvr`. If an optional `+` is appended to the type name, all processes that match this line spec will be combined into a single line.

name — Identifies the specific process. You can use a pattern identifier.

stat — A valid statistic name.

scale — A number.

filter — One of the following: `none`, `persecond`, `persample`, or `aggregate`.

axis — Either `y` or `y2`.

statFilter (optional)— a list that lets you exclude lines from the chart, based on the values of their statistical counters. Its format is:

`{count min max mean stddev}`

Any `statFilter` item can be an empty string {}, a single number, or a list of two numbers.

If an item is an empty string (that is, {}) it is ignored.

If an item is a list of two numbers (i.e. {36 100}) then the first number must be less than the second number. If the stat is less than the first or greater than the second, then it is excluded.

If an item is a single number then it is used as follows:

- If `count >= 0` then any line with fewer samples than the filter is excluded.
- If `count < 0` then any line with more samples than the absolute value of the filter is excluded.
- If `min >= 0` then any line whose min is less than the filter is excluded.
- If `min < 0` then any line whose min is greater than the filter is excluded.
- If `max >= 0` then any line whose max is less than the filter is excluded.
- If `max < 0` then any line whose max is greater than the absolute value of the filter is excluded.
- If `mean >= 0` then any line whose mean is less than the filter is excluded.
- If `mean < 0` then any line whose mean is greater than the absolute value of the filter is excluded.
- If `stddev >= 0` then any line whose stddev is less than the filter is excluded.
- If `stddev < 0` then any line whose stddev is greater than the absolute value of the filter is excluded.

statFilter examples

`{100}`

Only include stats that have at least 100 samples.

`{ } { 100 }`

Only include stats whose max is `>= 100`.

`{ } 20 { } { } -5 }`

Only include stats whose min `>= 20` and whose standard deviation is `< 5`.

`{{100 200} {} {-500} {120 240}}`

Only include stats that have at least 100 samples but no more than 200 samples; and whose maximum value is less than 500; and whose average value is between 120 and 240.

—
|

A

- AbortCount (cache statistics) 8-23
- abortErrLostOtRoot A-28
- abortFullBackup (Repository) 9-12
- accounts
 - administering user 5-16
- ad hoc processes 3-14, B-13
- addGroup:
 - (UserProfile) 5-36
- adding
 - a new user 5-17, 5-30
 - a user to a group 5-23, 5-36
 - user privileges 5-20, 5-23, 5-33
- addNewUserWithId:password:
 - (UserProfileSet) 5-30
- addNewUserWithId:password:...inGroups:
 - (UserProfileSet) 5-30
- addPrivilege:
 - (UserProfile) 5-33
- addTransactionLog:size: (Repository)
 - 7-10
- Admin GcGem
 - and GemStone privilege 5-6
 - and Reclaim GcGems 10-8
 - configuring A-7
 - defined 1-3
 - log files 4-19, 8-3, 8-4
 - running 10-29
- adminGcGemSessionId (System) 10-30
- administrative accounts 5-2
- administrative tools 5-1
 - GemBuilder for Smalltalk 5-11
 - Topaz 5-27
- AIO page server
 - defined 1-3
 - log files 4-19, 8-3, 8-5
- AioCkptCount (cache statistic) 8-23
- AioDirtyCount (cache statistic) 8-23
- AioNumBuffers (cache statistic) 8-23
- AioNumEmptyBuffers (cache statistic) 8-23
- AioRateLimit (cache statistics) 8-24

- AioRateMax (cache statistics) 8-24
 - AllClusterBuckets (predefined system object) defined D-5
 - AllGroups (predefined system object)
 - adding a user group to 5-37
 - adding a UserGroup to 5-25, 5-40
 - defined D-5
 - allocation
 - of extents, weighted 1-23
 - of objects to new extents 6-8, 6-9
 - of space for extent files A-12
 - AllUsers (predefined system object)
 - adding a UserProfile to 5-17, 5-30
 - adding to, GemBuilder 5-17
 - defined D-5
 - removing a UserProfile from 5-19
 - application
 - linked 2-2, 3-4
 - RPC (remote procedure call) 3-4
 - assigning privileges to a user 5-20, 5-30, 5-33
 - AsyncFlushesInProgress (cache statistics) 8-24
 - asynchronous I/O page server 1-44
 - audit report, example 8-16
 - auditWithLimit: (Repository) 8-12
 - auditWithLimit:reclaimAll: (Repository) 8-12
 - auditWithLimit:reclaimAll: (Repository) 8-13
 - auth modifier, NRS 3-13
 - authorization
 - and Segments 5-4, 5-23, 5-39
 - list of a Segment, removing a group from 5-25, 5-41
 - of a Segment, changing 5-25, 5-40
 - automatic garbage collection 10-11–10-25
 - #autoRefreshGcGemConfig 10-43
 - availability, maximizing 9-41
- B**
- backup
 - offline extent 9-33
 - restoring from 9-33
 - online extent 9-3
 - restoring from 9-7
 - backup and restore
 - and GemStone privilege 5-6
 - BackupLog 9-14
 - BackupRecordPagesWrittenByGem (cache statistic) 8-24
 - BackupRecordPagesWrittenByStone (cache statistic) 8-24
 - backups
 - checkpoint at start of 9-8
 - compressed 9-13
 - creating multi-part 9-12
 - examining internal fileId B-4
 - interactions with garbage collection 9-11
 - log of backups 9-14
 - managing a warm standby 9-41
 - of repository 9-2
 - of transaction logs 7-8, 9-2
 - on remote node 9-11
 - restoring 9-14
 - from multiple files 9-23
 - from offline extent backup 9-33
 - from tape 9-24
 - performance tips 9-23
 - to point in time 9-25
 - running a duplicate repository 1-48
 - transaction mode and 9-10
 - using copydbf 9-33
 - using operating system facilities 9-33
 - verifying readability 9-14
 - with repository online 9-8
 - beginTransaction (System) 4-25
 - BitlistPagesWrittenByGem (cache statistic) 8-24
 - BitlistPagesWrittenByStone (cache statistic) 8-24
 - BitmapPageReads (cache statistic) 8-24

- BmCHeapPages (cache statistic) 8-24
 - BmInternalPagesWrittenByGem (cache statistic) 8-24
 - BmInternalPagesWrittenByStone (cache statistic) 8-24
 - BmLeafPagesWrittenByGem (cache statistic) 8-25
 - BmLeafPagesWrittenByStone (cache statistic) 8-25
- C**
- cache statistics
 - about page reclamation 10-20, 10-44
 - description of all 8-23–8-61
 - filters for, in VSD F-13
 - monitoring 8-20
 - real-time monitoring F-5
 - user defined session 8-21
 - user-defined global session 8-22
 - _cacheName: (System) 8-21
 - cacheStatistics: (System) 8-20, 10-20, 10-44
 - cacheStatisticsDescription (System) 8-20
 - cacheStatisticsForSessionId: (System) 8-20
 - cacheStatsForGemWithName: (System) 8-20
 - calling C routines from Smalltalk 2-12
 - captive account mode, NetLDI
 - guest mode with 3-14
 - changing
 - a user's default segment 5-26, 5-41
 - the authorization of a Segment 5-25, 5-40
 - CHeapSizeKB (cache statistic) 8-25
 - checkpoint
 - defined 6-1
 - frequency of 1-42
 - identifying in transaction logs B-5
 - operating system backup and 9-33
 - Stone shutdown and 1-43
 - CheckpointCount (cache statistic) 8-25
 - checkpoints
 - starting 7-12
 - suspending 9-3
 - CheckpointState (cache statistic) 8-25
 - checkpointStatus (Repository) 9-3
 - checkpointStatus (System) 9-5
 - ClassesRead (cache statistic) 8-26
 - clearEpochGcState (System) 10-19
 - ClientLostOtsSent (cache statistics) 8-26
 - ClientPageReads (cache statistic) 8-26
 - ClientPageWrites (cache statistic) 8-26
 - ClientPid (cache statistics) 8-26
 - ClientSigAbortsSent (cache statistics) 8-26
 - clock, system 1-14
 - ClockHandFrameId (cache statistic) 8-26
 - cluster buckets
 - and AllClusterBuckets system object D-5
 - clustering, new extents and 6-8, 6-10
 - clustering, restoring backups and 9-16
 - code
 - region of temporary object memory 10-11
 - code modification
 - and GemStone privilege 5-6, 5-7
 - CodeCacheSizeBytes (cache statistic) 8-26
 - CodeGenGcCount (cache statistic) 8-26
 - commands
 - copydbf** B-2
 - gsl** B-7
 - pageaudit** B-9
 - removedbf** B-10
 - startcachewarmer** 4-23, B-11
 - startnetldi** B-13
 - startstone** B-15
 - when restoring from backups 9-7
 - stopnetldi** B-17
 - stopstone** B-18
 - topaz** B-19
 - waitstone** B-20
 - commit record
 - defined 10-3
 - commit record backlog 4-24, A-35
 - and problems with page reclamation

- 10-41
- defined 10-3
- CommitCount (cache statistic) 8-26
- CommitQueueAddedToRunQueueCount (cache statistic) 8-27
- CommitQueueAddedToRunQueueSessionCount (cache statistic) 8-27
- CommitQueueSessionNotReadyCount (cache statistic) 8-27
- CommitQueueSize (cache statistic) 8-27
- CommitQueueThreshold (cache statistic) 8-27
- CommitRecordCount (cache statistic) 8-27
- CommitRecordDisposalsDeferredCount (cache statistic) 8-27
- CommitRecordDisposalState (cache statistic) 8-27
- CommitRecordPagesWrittenByGem (cache statistic) 8-28
- CommitRecordPagesWrittenByStone (cache statistic) 8-28
- CommitRecordsDisposable (cache statistic) 8-28
- CommitRecordsDisposedCount (cache statistics) 8-28
- CommitRecordsReadAbortCount (cache statistics) 8-28
- CommitRecordsReadCommitWithoutTokenCount (cache statistics) 8-28
- CommitRecordsReadCommitWithTokenCount (cache statistics) 8-28
- commitRestore (Repository) 9-7, 9-22, 9-37 and GemStone privilege 5-6
- CommitRetryFailureCount (cache statistic) 8-28
- CommitsSinceLastEpoch (cache statistic) 8-28
- CommitTokenSession (cache statistic) 8-28
- communications, disrupted 3-7
- compiler and symbol resolution 5-8
- compressed backups
 - creating 9-13
- configuration
 - access at run time
 - Gem 2-9
 - Stone 1-38
 - extent locations 1-20
 - file descriptors 1-19
 - Gem session processes 2-4
 - kernel resources 1-14
 - memory needs for server 1-12
 - multiple extents 1-23
 - raw partitions 1-34
 - shared page cache 1-15
 - single-host 2-2
 - Stone private page cache 1-17
 - swap space needs 1-13
 - system resources 1-12
 - transaction logs 1-27
 - tuning 1-41
 - visual statistics display F-16
- configuration files
 - examining parameters from Smalltalk 1-38, 2-9
 - executable A-1
 - for server (Stone) 1-4
 - for sessions 2-3
 - for visual statistics display F-16
 - naming options A-9
 - option value errors in A-11
 - options, warning messages and A-9
 - printing summary of all options A-14
 - searching for A-2
 - specific to GcGems A-7
 - syntax errors in A-11
 - syntax of A-9
 - system-wide A-1
 - Topaz and A-8
 - used by GemStone A-2
- configuration options
 - and ConfigurationParameterDict system

object D-5
DBF_ALLOCATION_MODE 1-23, A-12
DBF_EXTENT_NAMES 1-20, 4-6, A-12
 and Reclaim GcGems 10-36
DBF_EXTENT_SIZES 1-20, 1-23, 6-15,
 A-13
DBF_PRE_GROW 1-22, A-13
DBF_SCRATCH_DIR A-14
DUMP_OPTIONS A-14
for visual statistics display F-16
GEM_FREE_FRAME_CACHE A-14
GEM_FREE_FRAME_CACHE_SIZE 1-45
GEM_FREE_FRAME_LIMIT 1-44, 2-10,
 A-14
GEM_GCI_LOG_ENABLED A-15
GEM_HALT_ON_ERROR A-15
GEM_IO_LIMIT 2-10, A-15
GEM_KEEP_MIN_SOFTREFS A-16
GEM_PGSRV_FREE_FRAME_CACHE_SIZE 1-45, A-16
GEM_PGSRV_FREE_FRAME_LIMIT A-17
GEM_PGSRV_UPDATE_CACHE_ON_READ 2-10, 3-2, A-17
GEM_PRIVATE_PAGE_CACHE_KB 2-12,
 A-17
GEM_RPCGCI_TIMEOUT A-18
GEM_SEND_STN_MSGS_VIA_PGSRV A-18
GEM_SMALLTALK_STACK_DEPTH A-16
GEM_SOFTREF_CLEANUP_PERCENT_MEMORY A-18
GEM_TEMPOBJ_AGGRESSIVE_STUBBING A-19
GEM_TEMPOBJ_CACHE_SIZE 2-11, 10-14,
 A-19
GEM_TEMPOBJ_INITIAL_SIZE 10-14,
 A-20
GEM_TEMPOBJ_MESPACE_SIZE 10-14,
 A-20
GEM_TEMPOBJ_OOPMAP_SIZE 10-14,
 A-20
GEM_TEMPOBJ_POMGEN_SIZE 10-14,
 A-21
KEYFILE A-21
LOG_WARNINGS A-21
SHR_NUM_FREE_FRAME_SERVERS 1-45,
 A-22
SHR_PAGE_CACHE_LOCKED A-22
SHR_PAGE_CACHE_NUM_PROCS 2-6,
 A-22
SHR_PAGE_CACHE_NUM_SHARED_COUNTERS A-22
SHR_PAGE_CACHE_SIZE_KB 1-17, 2-6,
 A-23
SHR_SPIN_LOCK_COUNT 1-41, A-23
SHR_TARGET_FREE_FRAME_COUNT A-24
STN_ADMIN_GC_SESSION_ENABLED 10-8, 10-29, A-24
STN_ALLOCATE_HIGH_OOPS A-24
STN_CHECKPOINT_INTERVAL 1-43, A-25
STN_COMMIT_QUEUE_THRESHOLD A-25
STN_COMMITS_ASYNC A-26
STN_COMMIT_TOKEN_TIMEOUT A-25
STN_CR_BACKLOG_THRESHOLD A-26
STN_DISABLE_LOGIN_FAILURE_LIMIT 5-49, A-26
STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT 5-49, A-26
STN_DISKFULL_TERMINATION_INTERVAL 6-15, A-27
STN_EPOCH_GC_ENABLED 10-18, A-27
STN_FREE_FRAME_CACHE_SIZE A-27
STN_FREE_SPACE_THRESHOLD 6-14,
 A-27
STN_GEM_ABORT_TIMEOUT A-28
STN_GEM_LOSTOT_TIMEOUT A-28
STN_GEM_TIMEOUT A-29
STN_HALT_ON_FATAL_ERR 1-30, A-29
STN_LOG_LOGIN_FAILURE_LIMIT 5-49, A-30
STN_LOG_LOGIN_FAILURE_TIME_LIMIT

- IT 5-49, A-30
- STN_LOOP_NO_WORK_THRESHOLD A-30
- STN_MAX_AIO_RATE A-31
- STN_MAX_AIO_REQUESTS A-31
- STN_MAX_SESSIONS 1-18, A-32
- STN_MAX_VOTING_SESSIONS A-32
- STN_NUM_GC_RECLAIM_SESSIONS
10-8, 10-36, A-32
- STN_NUM_LOCAL_AIO_SERVERS 1-44,
A-33
- STN_OBJ_LOCK_TIMEOUT A-33
- STN_PAGE_REMOVAL_THRESHOLD A-33
- STN_PRIVATE_PAGE_CACHE_KB 1-17,
A-34
- STN_REMOTE_CACHE_TIMEOUT A-34
- STN_SHR_TARGET_PERCENT_DIRTY
A-34
- STN_SIGNAL_ABORT_CR_BACKLOG
1-42, A-35
- STN_TRAN_FULL_LOGGING 1-28, 7-3,
A-35
- STN_TRAN_LOG_DEBUG_LEVEL A-36
- STN_TRAN_LOG_DIRECTORIES 1-30,
1-47, A-36
- STN_TRAN_LOG_LIMIT 1-43, A-36
- STN_TRAN_LOG_PREFIX A-36
- STN_TRAN_LOG_SIZES 1-30, A-37
- STN_TRAN_Q_TO_RUN_Q_THRESHOLD
A-37
- configuration parameters
 - garbage collection 10-42
 - Reclaim GcGem 10-42
- configurationAt: (System) 1-38, 2-9, A-37
- configurationAt: put: (System) 1-39,
2-9
- ConfigurationParameterDict (predefined
system object)
 - defined D-5
- configurations, typical 1-5
- configuring
 - Reclaim GcGems 10-36
- continueFullBackupCompressedTo:MB
 - ytes
(Repository) 9-13
- continueFullBackupTo:MBytes
(Repository) 9-12
- copydbf** command
 - archiving transaction logs 7-8
 - description B-2
 - example 3-16
 - with raw partition 1-35
 - using with raw partitions 1-35
- CountBagInteriorPagesWrittenByGem (cache
statistic) 8-29
- CountBagInteriorPagesWrittenByStone
(cache statistic) 8-29
- CountBagLeafPagesWrittenByGem (cache
statistic) 8-29
- CountBagLeafPagesWrittenByStone (cache
statistic) 8-29
- CountMapLeafPagesWrittenByGem (cache
statistic) 8-29
- CountMapLeafPagesWrittenByStone (cache
statistic) 8-29
- createExtent: (Repository) 6-6
- createExtent:withMaxSize:
(Repository) 6-7
- creating
 - executable configuration files A-6
 - extents 1-27
 - new user group 5-23, 5-36
 - new UserProfile 5-17, 5-30
 - transaction logs 1-27, 7-3
- current Segment, exercise caution when
changing 5-40
- currentGcReclaimSessionsByExtent
(System) 10-38, 10-40
- currentLogDirectoryId (Repository)
7-10
- currentLogFile (Repository) 7-10
- currentSessionNames (System) 4-16, 4-17,
10-41
 - and GemStone privilege 5-6
- currentTranlogSizeMB (Repository) 7-9,
7-10

D

- data curator
 - and DataCuratorSegment object D-4
- DataCurator
 - and AllUsers system object D-5
 - and DataCuratorSegment object D-4
 - defined D-2
 - described 5-7
 - logging in as 5-2
 - tasks D-2
- DataCuratorSegment (predefined system object)
 - defined D-4
- #dataPageBufferSize 10-43
- DataPageReads (cache statistic) 8-29
- DBF_ALLOCATION_MODE
 - (configuration option) 1-23, A-12
 - adding extents and 6-8, 6-9
 - effect when restoring backups 9-16, 9-23
- DBF_EXTENT_NAMES
 - (configuration option) 4-6, A-12
 - effect when restoring backups 9-17, 9-34
- DBF_EXTENT_NAMES (configuration option)
 - 6-5, 6-7, 6-8, 6-9
 - and Reclaim GcGems 10-36
- DBF_EXTENT_SIZES (configuration option)
 - 1-23, 6-5, 6-15, 7-1, A-13
- DbfHistory (predefined system object)
 - defined D-4
- DBF_PRE_GROW (configuration option) 1-23, 6-6, 6-7, A-13
- DBF_SCRATCH_DIR (configuration option)
 - A-14
- dead object
 - contrasted with shadow object 10-3–10-6
 - defined 10-3
- DeadDataPagesWrittenByGem (cache statistic) 8-29
- DeadDataPagesWrittenByStone (cache statistic) 8-29
- DeadNotReclaimedKobjs (cache statistic) 8-29
 - and reclamation process 10-43
- DeadObjsReclaimedCount (cache statistic)
 - 8-29
 - and reclamation process 10-43
- DecimalMinusInfinity (Float constant)
 - defined D-3
- DecimalMinusQuietNaN (Float constant)
 - defined D-3
- DecimalMinusSignalingNaN (Float constant)
 - defined D-3
- DecimalPlusInfinity (Float constant)
 - defined D-3
- DecimalPlusQuietNaN (Float constant)
 - defined D-3
- DecimalPlusSignalingNaN (Float constant)
 - defined D-3
- default
 - NetLDI mode 3-9
- default name
 - of NetLDI process 3-4
- default scratch directories A-14
- default segment
 - changing a user's 5-26, 5-41
 - defined 5-4
 - exercise caution when changing 5-40
 - privilege required in UserProfile 5-26, 5-41
 - specifying for a new user 5-26
- default system-wide configuration files A-12
- defaultSegment: (UserProfile) 5-41
- DeferCkptCompleteCount (cache statistic)
 - 8-30
- #deferReclaimCacheDirtyThreshold
 - GcUser parameter 10-43
- deletePrivilege: (UserProfile) 5-34
- deleting user 5-20
- deleting user privileges 5-20, 5-34
- DepMapKeysChanged (cache statistic) 8-30
- dereferencing large objects in repository 10-46
- descriptionOfSession (System) 4-17
- descriptionOfSession: (System) 10-41

- dictionaries
 - adding to symbol list 5-21
 - Globals 5-9
 - Published 5-9
 - UserGlobals 5-9
- directory, current, of child process 3-16
- DirtyListSize (cache statistic) 8-30
- DirtyPageSweepCount (cache statistic) 8-30
- disableEpochGc (System) 10-18
- disabling logins 9-18
- disaster recovery B-16
- disk drives
 - I/O among multiple extents 1-24
 - multiple drives recommended 1-9
 - raw partitions 1-34
 - usage recommendations 1-9
- disk failure 4-20
- disk or repository full error 6-14
- disk space
 - disk-full errors 6-14
- DUMP_OPTIONS (configuration option) A-14
- E**
- EmptyBmLeafPagesWrittenByGem (cache statistic) 8-30
- EmptyBmLeafPagesWrittenByStone (cache statistic) 8-30
- #enableDebugging 10-43
- enableEpochGc (System) 10-18
- environment variables 5-18, E-1
 - GEMBUILDER 5-18
 - GEMSTONE_EXE_CONF 5-18, A-1, A-5, A-7, A-8
 - GEMSTONE_LOG 5-18, 8-3
 - GEMSTONE_MAX_FD 1-14, 2-5
 - GEMSTONE_NRS_ALL 3-16, 5-18
 - GEMSTONE_SYS_CONF 5-18, A-1, A-3, A-12
 - reserved names E-5
 - used in options A-10
- epoch garbage collection 10-10, 10-17–10-25
 - benefits of 10-17
 - determining epoch length 10-20
 - disabling 10-18
 - enabling 10-18
 - forcing 10-19
 - limitations of 10-18
 - privilege required 10-17
 - resetting the state of 10-19
 - when 10-21
- EpochGcCount (cache statistic)
 - and epoch garbage collection 10-20
- #epochGcTimeLimit 10-20
- #epochGcTransLimit 10-20
- EpochNewKobjs (cache statistic)
 - and epoch garbage collection 10-20
- EpochPossibleDeadKobjs (cache statistic)
 - and epoch garbage collection 10-20
- EpochScannedKobjs (cache statistic)
 - and epoch garbage collection 10-20
- error messages
 - native language D-4
 - presented in the user's native language 5-9
- error numbers 4-4, 4-9

- errors
 - after restoring backup 9-16
 - disk full, diagnosing 6-14
 - error numbers 4-4
 - extent already exists 4-6
 - extent already open 4-5
 - extent missing or access denied 4-5
 - fatal 1-30, A-29
 - in configuration files, option value A-11
 - in configuration files, syntax A-11
 - invalid password 5-50
 - key file 4-4
 - object audit 8-15
 - restoring transaction logs 9-26
 - shared page cache not attached 4-4
 - SigLostOTRoot A-28
 - Stone response to Gem fatal error 1-30
 - stuck spin lock 4-21
 - tranlog directories full 7-13
 - transaction log missing 4-7
 - ErrorSymbols (predefined system object)
 - defined D-6
 - /etc/services file 3-4, 3-18
 - examining
 - user group memberships 5-23, 5-36
 - user privileges 5-5, 5-20, 5-32
 - executable configuration files A-8, A-12
 - creating A-6
 - defined A-1
 - names A-7
 - search for A-5
 - setting permission A-7
 - expanding extents 6-15
 - ExportedSetSize (cache statistic) 8-31
 - extent files 6-15
 - see also *repository*
 - allocating space A-12
 - allocation mode 1-23
 - checkpoint defined 6-1
 - creating new 1-27, 6-6
 - defined 1-3
 - disk full condition 6-14
 - disk space, managing 6-15
 - estimating size of 1-19
 - examining internal fileId B-4
 - free space in 6-3
 - group access to 1-33
 - identifying an extent B-4
 - location 1-20
 - maximum size 1-21, A-13
 - moving to raw partition 1-36
 - naming A-12
 - permissions for dynamically added 6-5
 - pre-growing 1-21, A-13
 - reallocating objects in 6-8, 6-9
 - recovery after file system repair 9-37
 - removing 6-7
 - size 1-19
 - specifying size of A-13
 - using multiple extents 1-23
 - ExtentFlushCount (cache statistics) 8-31
- F**
- failed login messages in log 5-49
 - FailedCommitCount (cache statistic) 8-31
 - false (predefined system object)
 - defined D-3
 - fatal errors A-29
 - FDC
 - and MGC 10-10
 - and MGC (garbage collection) 10-31
 - fast 10-33
 - file descriptors 1-13, 1-19, 2-5
 - file permissions 1-31
 - fileId, of repository files B-4

- files
 - ..LCK lock files 3-4
 - permissions for Gem processes 2-7
 - fileSize (Repository) 6-3
 - fileSizeReport (Repository) 6-3
 - findDisconnectedObjectsAndWriteToFile: (Repository) 10-10, 10-31
 - progress count during 8-50
 - findObjectsLargerThan:limit:(Object) 10-44
 - findReferences: (Object) 10-45
 - findReferencesWithLimit: (Object) 10-45
 - forceEpochGc (System) 10-19
 - fork-in-time scenario 9-31
 - FragmentBmPagesWrittenByGem (cache statistic) 8-32
 - FragmentBmPagesWrittenByStone (cache statistic) 8-32
 - FrameCount (cache statistic) 8-32
 - FramesFromFindFree (cache statistic)
 - for evaluating whether free frame page server is needed 1-45
 - FramesAddedToFreeList (cache statistic) 8-32
 - for evaluating whether free frame page server is needed 1-45
 - FramesFromFindFree (cache statistic) 8-32
 - FramesFromFreeList (cache statistic) 8-32
 - free frame cache
 - specifying the size of A-14
 - Free Frame page server
 - log files 4-19
 - free frame page server 1-44
 - defined 1-3
 - log files 8-3, 8-5
 - free list page server
 - benefits of 1-44
 - free space
 - in repository 6-3
 - FreeFrameCacheNumFrames (cache statistic) 8-32
 - FreeFrameCacheSize (cache statistic) 8-32
 - FreeFrameCount (cache statistic) 8-32
 - for evaluating whether free frame page server is needed 1-45
 - FreeFrameLimit (cache statistic) 8-32
 - for evaluating whether free frame page server is needed 1-45
 - freeing repository space
 - when 10-8
 - FreeOopsK (cache statistic) 8-33
 - FreePages (cache statistic) 8-33
 - freeSpace (Repository) 6-3
 - full backup 9-2
 - full logging
 - how to manage 7-7
 - fullBackupCompressedTo: (Repository) 9-13
 - fullBackupCompressedTo:MBytes: (Repository) 9-13
 - fullBackupTo: (Repository) 9-8, 9-9
 - progress count during 8-50
 - fullBackupTo:MBytes: (Repository) 9-8, 9-9
- G**
- garbage collection
 - automatic mechanisms 10-11–10-25
 - automatic vs. invoked 10-9
 - #autoRefreshGcGemConfig 10-43
 - backups and 9-11
 - collecting information on 10-43
 - commit record backlog, defined 10-3
 - commit record, defined 10-3
 - concepts 10-1
 - configuring GcGems specially A-7
 - conflicts between mechanisms 10-27, 10-34
 - #dataPageBufferSize 10-43
 - dead object, defined 10-3
 - #deferReclaimCacheDirtyThresho

ld 10-43
determining epoch length 10-20
#enableDebugging 10-43
epoch collection 10-17–10-25
 when 10-21
#epochGcTimeLimit 10-20
#epochGcTransLimit 10-20
from local object memory 10-11
GcUser configuration parameters 10-42
identifying garbage 10-2
live object, defined 10-2
local object memory 10-9
markForCollection Repository
 method 10-26
mark/sweep, defined 10-6
#maxTransactionDuration 10-42
#mfcPageBufSize 10-28
object table sweep, defined 10-7
#objsMovedPerCommitThreshold
 10-42
overview 10-2
pages, defined 10-9
parallel reclaim 10-10
possible dead objects, defined 10-7
process overview 10-6–10-8
#reclaimDeadEnabled 10-42
#reclaimDeadShadowPageThreshol
 d 10-43
reclaiming 10-2
#reclaimMinPages 10-42
#reclaimSleepTime 10-42
resources reclaimed 10-2
shadow object, defined 10-3
#sleepTimeBetweenReclaim 10-42
Smalltalk methods that require explicit
 privilege 5-6
#StnAdminGcSessionEnabled 10-29
targeted marking 10-10
transitive closure, defined 10-2
tuning reclaim 10-42, 10-43
two-step process 10-31
#verboseLogging 10-43
voting, defined 10-7
write set union sweep, defined 10-7
GcGems
 special configuration file for A-7
 tasks of 10-8
GciRpcCommandsServiced (cache statistic)
 8-33
GcUser
 changing parameters for 10-42
 defined D-2
 described 5-7
 detecting changed parameters 10-42
 logging in as 5-3
GcUser parameter 10-42, 10-43
GcVoteState (cache statistic) 8-33
 and reclamation process 10-44
GcWsUnionSweepCount (cache statistic) 8-33
Gem
 custom executable 3-6

- Gem session process
 - configuration 2-4
 - file 2-3
 - run time access to 2-9
 - tuning 2-11
 - configuring 2-6
 - custom executable, installing 2-12
 - defined 2-2
 - file ownership and permissions for 2-6
 - linked and RPC 2-3
 - linked, setting up access 2-7
 - log files related to 8-7
 - private page cache, setting size of A-17
 - remote from stone 3-18
 - RPC or remote, setting up access 2-8
 - starting 4-10
 - linked session 4-11
 - RPC session 4-13
 - troubleshooting 4-14
 - swapping of, excessive 2-12
 - system resources for 2-4
 - temporary object space, tuning 2-11
 - tuning configuration 2-11
- GemBuilder
 - adding users 5-17
 - privileges, modifying 5-20
 - removing users 5-19
 - repository protection and 1-33
 - S bit and 2-8
 - Segment tool 5-24, 5-25
- GEMBUILDER (environment variable) 5-18
- GemBuilder for Smalltalk
 - used for system administration 5-11
- gem.conf file A-7
- gemConfigurationAt: (System) 2-9
- gemConfigurationReport (System) 2-9
- GEM_FREE_FRAME_CACHE
 - (configuration option) A-14
- GEM_FREE_FRAME_CACHE_SIZE
 - (configuration option) 1-45
- GEM_FREE_FRAME_LIMIT (configuration option) 1-44, 2-10, A-14
- GemFreeFrameLimit (internal parameter) 2-10, A-14
- GemFreePages (cache statistic) 8-34
- GEM_GCI_LOG_ENABLED (configuration option) A-15
- GEM_HALT_ON_ERROR (configuration option) A-15
- GemHasCommitToken (cache statistic) 8-34
- GEM_IO_LIMIT (configuration option) 2-10, A-15
- GemIOLimit (internal parameter) 2-10, A-15
- GEM_KEEP_MIN_SOFTREFS (configuration option) A-16
- GEM_MAX_SMALLTALK_STACK_DEPTH (configuration option) A-16
- GemNetId for remote Stone 3-25
- gemnetobject executable
 - for custom Gem executable 2-13
 - mapping 3-6
 - modifying for custom Gem executable 2-12
 - RPC session and 4-14
- GEM_PGSRV_FREE_FRAME_CACHE_SIZE (configuration option) A-16
- GEM_PGSRV_FREE_FRAME_CACHE_SIZE (configuration option) 1-45
- GEM_PGSRV_FREE_FRAME_LIMIT (configuration option) A-17
- GEM_PGSRV_UPDATE_CACHE_ON_READ (configuration option) 2-10, 3-2, A-17
- GemPgsvrUpdateCacheOnRead (internal parameter) 2-10, A-17
- GEM_PRIVATE_PAGE_CACHE_KB (configuration option) A-17
- tuning 2-12
- GEM_RPCGCI_TIMEOUT (configuration option) A-18
- GEM_SEND_STN_MSGS_VIA_PGSRV (configuration option) A-18
- gemsetup.csh
 - example 3-22, 4-3, 4-9
- gemsetup.sh
 - example 3-22, 4-3, 4-9
- GemsInCacheCount (cache statistic) 8-34

- GEM_SOFTREF_CLEANUP_PERCENT_MEM
(configuration option) A-18
- GemStone
see also *Stone repository monitor* and *Gem session process*
actions, user-defined 2-12
adding user privileges 5-4, 5-20, 5-33
component overview 2-1
configuration files used in A-2
examining user privileges 5-5, 5-20, 5-32
modifying another user's ID 5-5, 5-38
network configuration and installation
3-2
password, defined 5-4
password, modifying another user's 5-20,
5-32
password, modifying your own 5-20, 5-31
privileges required for system
administration tasks 5-2
privileges, defined 5-4
redefining user privileges 5-5, 5-34
removing user privileges 5-20, 5-34
service name 2-12
shutting down repository 4-17
avoid **kill -9** 4-18
starting repository monitor 4-2
SymbolDictionaries, used in symbol
resolution 5-8
system logs, examining 8-2
typical configurations 1-5
user ID, defined 5-3
users, access to network 3-13
- GEMSTONE (environment variable)
setting 4-11
- GemStone repository
initial contents D-1
- GemStoneError (predefined system object)
defined D-6
- GEMSTONE_EXE_CONF (environment
variable) 4-11, 5-18, A-1, A-5, A-6,
A-7, A-8
- GEMSTONE_LOG (environment variable) 5-18,
8-3
- GEMSTONE_MAX_FD (environment variable)
1-14, 2-5
- GEMSTONE_NRS_ALL (environment variable)
3-5, 3-16, 5-18
- GEMSTONE_SYS_CONF (environment
variable) 5-18, A-1, A-2, A-6, A-12
- GEM_TEMPOBJ_AGGRESSIVE_STUBBING
(configuration option) A-19
- GEM_TEMPOBJ_CACHE_SIZE (configuration
option) 10-14, A-19
and bulk loading of objects 4-22
tuning 2-11
- GEM_TEMPOBJ_INITIAL_SIZE
(configuration option) 10-14, A-20
- GEM_TEMPOBJ_MESPACE_SIZE
(configuration option) 10-14, A-20
- GEM_TEMPOBJ_OOPMAP_SIZE
(configuration option) 10-14, A-20
- GEM_TEMPOBJ_POMGEN_SIZE
(configuration option) 10-14, A-21
- global session statistics 8-22, 8-34
- GlobalDirtyPageCount (cache statistic) 8-34
- Globals (system globals dictionary) 5-9
initial contents of D-2, D-6
- globalSessionStatAt
put: (System) 8-22
- globalSessionStatAt: (System) 8-22
- GlobalStatn 8-34
- group:authorization: (Segment) 5-25,
5-40, 5-41
- groups
access to extents 1-33
adding a new user to 5-30, 5-36
and AllGroups system object D-5
and Segment authorization 5-4
and Segment authorization, GemBuilder
5-24, 5-25
creating new 5-37
defined 5-5
examining a user's memberships 5-23,

5-36
GemBuilder Admin Tool and 5-24, 5-25
list all members of 5-37
removing 5-38
removing a user from 5-23, 5-37
removing from a Segment's authorization list 5-25, 5-41
groups (UserProfile) 5-36
GS_CORE_TIME_OUT (environment variable) E-2
GS_DEBUG_VMGC_MKSW_MEMORY_USED_SOFT_BREAK (environment variable) 10-12, E-3
GS_DEBUG_VMGC_MKSW_PRINT_C_STACK (environment variable) 10-12, E-3
GS_DEBUG_VMGC_MKSW_PRINT_STACK (environment variable) 10-12, E-3
GS_DEBUG_VMGC_PRINT_MKSW (environment variable) 10-13, E-3
GS_DEBUG_VMGC_PRINT_MKSW_MEMORY (environment variable) 10-13, E-3
GS_DEBUG_VMGC_PRINT_MKSW_MEMORY_USED (environment variable) 10-13, E-3
GS_DEBUG_VMGC_PRINT_SCAV (environment variable) 10-13, E-3
GS_DEBUG_VMGC_PRINT_TRANS (environment variable) E-4
GS_DEBUG_VMGC_SCAV_PRINT_C_STACK (environment variable) 10-13, E-4
GS_DEBUG_VMGC_SCAV_PRINT_STACK (environment variable) 10-13, E-4
GS_DEBUG_VMGC_VERBOSE_OUTOFMEM (environment variable) 10-13, E-4
GS_DEBUG_VMGC_VERIFY_MKSW (environment variable) 10-13, E-4
GS_DEBUG_VMGC_VERIFY_SCAV (environment variable) 10-14, E-4
GS_DEBUG_VM_PRINT_TRANS (environment variable) 10-13
GS_DISABLE_KEEPLIVE (environment variable) E-4
GS_DISABLE_WARNING (environment variable) E-4

gslist command
 description B-7
 executable 4-10
 finding log locations 8-2
GsMsgCount (cache statistic) 8-34
GsMsgKind (cache statistic) 8-34
GsMsgSessionId (cache statistic) 8-34
GS_WRITE_CORE_FILE (environment variable) E-4
guest mode, NetLDI 3-10
 captive accounts with 3-14

H

hasMissingGcGems (System) 10-30, 10-40

I

identifying garbage 10-2
identifying large objects in the repository 10-44
incrementGlobalSessionStatAt:by: (System) 8-23
insertDictionary:at: (UserProfile) 5-34
installing
 shared custom Gem executables 2-12
instance creation
 and **InstancesDisallowed** system object D-6
InstancesDisallowed (predefined system object)
 defined D-6
internal parameters A-37
invalid password error 5-50
InvalidPagesWrittenByGem (cache statistic) 8-35
InvalidPagesWrittenByStone (cache statistic) 8-35

K

keepalive, network option 3-7

- kernel requirements
 - for Gem session processes 2-6
 - for Stone repository monitor 1-14
- KEYFILE
 - (configuration option) A-21
- killing Gem or Stone processes 4-18
- L**
- large objects, identifying in the repository
 - 10-44
- LastSessionFatalError (cache statistic) 8-35
- LastSessionIdStopped (cache statistic) 8-35
- LastSessionIdTerminated (cache statistic) 8-35
- LastSessionLostOt (cache statistic) 8-35
- LastSessionSigAbort (cache statistic) 8-35
- LastWakeupInterval (cache statistic) 8-35
- LdiThreadOperations (cache statistic) 8-35
- libposix-aio.so B-15
- licensing keyfile
 - setting the location of A-21
- linked application 2-2, 3-4
- listing all members of a group 5-37
- live object
 - defined 10-2
- local object memory
 - defined 10-9
 - garbage collection of 10-9
- LocalCacheAllocatedPceCount (cache statistic) 8-35
- LocalCacheFreeFrameCount (cache statistic) 8-35
- LocalCacheFreePceCount (cache statistic) 8-36
- LocalCacheOverflowCount (cache statistic) 8-36
- LocalCachePceCountLimit (cache statistic) 8-36
- LocalCachePceReclaimCount (cache statistic) 8-36
- LocalCacheStalePcesRemovedCount (cache statistic) 8-36
- LocalCacheValidPcesRemovedCount (cache statistic) 8-36
- LocalDirtyPageCount (cache statistic) 8-36
- LocalPageCacheHits (cache statistic) 8-36
- LocalPageCacheMisses (cache statistic) 8-36
- LocalPageCacheWrites (cache statistic) 8-37
- Lock1WaitQueueSize (cache statistic) 8-37
- Lock2WaitQueueSize (cache statistic) 8-37
- LockReqQueueSize (cache statistic) 8-37
- log files 8-2–8-9
 - Admin GcGem 4-19, 8-4
 - AIO page server 4-19, 8-5
 - for child processes 2-8
 - for RPC Gems 2-8
 - free frame page server 4-19, 8-5
 - garbage collection session 4-19
 - Gem session process 8-7
 - NetLDI 8-9
 - Page Manager 4-19, 8-6
 - Reclaim GcGem 4-19, 8-6
 - shared page cache monitor 1-19, 4-19
 - Stone repository monitor 4-19
 - SymbolGem 4-19, 8-7
 - write access for 1-33, 2-8
- logging in, example 5-28
- LogHighQueueAdds (cache statistic) 8-37
- LogHighQueueSize (cache statistic) 8-37
- login segment, described 5-4
- LoginQueueSize (cache statistic) 8-37
- LoginRequestsCount (cache statistic) 8-37
- LogLowQueueAdds (cache statistic) 8-37
- LogLowQueueSize (cache statistic) 8-37
- LogOriginTime (internal parameter) A-38
- logOriginTime (Repository) 7-10
- LogRecordPagesWrittenByGem (cache statistic) 8-38
- LogRecordPagesWrittenByStone (cache statistic) 8-38
- LogRecordsIoCount (cache statistic) 8-38
- LogRecordsWritten (cache statistic) 8-38
- LogWaitQueueSize (cache statistic) 8-38
- LOG_WARNINGS (configuration option) A-21
- lostOt, default timeout A-28
- LostOtPagesWrittenByGem (cache statistic)

- 8-38
 LostOtPagesWrittenByStone (cache statistic) 8-38
 lostOTRoot timeout A-28
 LostOtsReceived (cache statistic) 8-38
 LostOtsSent (cache statistic) 8-38
- M**
- manual transaction mode 4-25, 8-1, 9-10
 markForCollection
 and rtErrSignalAbort 10-26
 conflicts with other garbage collection 10-27
 privilege required 10-26
 reducing impact on other sessions 10-28
 scheduling 10-28
 markForCollection (Repository) 10-10, 10-26
 and Admin GcGem 10-29
 and GemStone privilege 5-6
 progress count during 8-50
 size of mark/sweep buffer and 10-28
 markGcCandidatesFromFile:
 conflicts with other garbage collection 10-34
 privilege required 10-34
 markGcCandidatesFromFile:
 (Repository) 10-10, 10-31
 and Admin GcGem 10-29
 marking garbage repository-wide 10-10
 marking objects for garbage collection 10-26
 marking specific objects 10-10
 mark/sweep
 defined 10-6
 MarkSweepCount (cache statistic) 8-39
 10-42
 #maxTransactionDuration 10-42
 MaxVotingSessions (cache statistic) 8-39
 mE
 region of temporary object memory 10-11
 membersOfGroup:
 (UserProfileSet) 5-37
 memory
 Gem session processes 2-5
 server needs 1-12
 signalling on low 10-17
 MeSpaceAllocatedBytes (cache statistic) 8-39
 MeSpaceUsedBytes (cache statistic) 8-39
 MessageKindToStone (cache statistic) 8-39
 MessagesToStnProcessingCommit (cache statistic) 8-39
 MessagesToStnStoneCommit (cache statistic) 8-39
 MessagesToStnWaitingForCommit (cache statistic) 8-39
 MessagesToStone (cache statistic) 8-39
 MethodsRead (cache statistic) 8-39
 mfcGcPageBufSize 10-28
 MGC
 and FDC 10-10
 and FDC (garbage collection) 10-31
 MilliSecPerIoSample (cache statistic) 8-40
 MinusInfinity (Float constant)
 defined D-3
 MinusQuietNaN (Float constant)
 defined D-3
 MinusSignalingNaN (Float constant)
 defined D-3
 modes
 allocation 1-8, 1-23, A-12
 debugging (NetLDI) B-13
 file protection 1-32, 2-7
 full logging 1-3, 1-11, 1-28, 7-1, 7-3, 7-5, 9-14, A-35
 guest (NetLDI) B-13
 manual transaction 8-1, 9-10
 partial logging 1-29, 7-3, 7-13, 9-22
 transaction logging 1-27
 modifying
 another user's ID 5-38
 another user's password 5-20, 5-32
 your own password 5-20, 5-31

monitoring
 cache statistics 8-20
MultObjPagesWrittenByGem (cache statistic)
 8-40
MultObjPagesWrittenByStone (cache statistic)
 8-40
myCacheProcessSlot (System) 8-20

N

Nameless account 5-8, D-2
naming
 configuration file options A-9
 executable configuration files A-7
 extent files A-12
NativeLanguage (predefined System object)
 and UserGlobals dictionary D-2
 defined D-4
NativeLanguage (predefined system object)
 defined in UserGlobals 5-9
NetLDI
 log files 8-9
NetLDI (GemStone network server process)
 3-4
 captive account mode 3-10
 debug mode 3-33
 environment variable for 3-5
 list of 4-10
 permissions for 3-4, 3-12, 3-14
 shutting down 4-17
 starting 3-10, 4-8
 troubleshooting 4-9
NetLDI modes
 captive account 3-9, 3-14
 default 3-9
 guest 3-10, 3-14, 4-13
 secure 3-9
NetLDI process
 default name 3-4
netl did, *see* *NetLDI*
.netrc file 3-13

network
 authentication, when required 3-9
 configuration 3-2
 copying repository files across 3-16
 disrupted communications 3-7
 /etc/services file 3-4, 3-18
 Gem session process on Stone's machine
 3-23
 GemStone network objects (gemnetobject)
 3-5
 guest mode with captive account 3-14
 keepalive option 3-7
 linked application on remote machine
 3-20
 log file for NetLDI 8-9
 log files for spawned processes 2-8
 NetLDI 3-4
 objects, mapping to executables 3-5
 page server processes 3-5
 password authentication 3-12
 remote sessions, setting up 3-18
 resource string (NRS) 3-15
 syntax C-1
 setting up remote sessions 3-18
 shared GemStone directory 3-19
 shared page cache and 3-6
 shell scripts, modifying for custom Gem
 executable 2-12
 software, TCP/IP 3-5
 Stone and RPC Gem on different
 machines 3-25
 troubleshooting remote logins 3-30
 typical configurations 3-2
network resource string
 #auth modifier 3-13
NetWriteThreadSocketWrites (cache statistic)
 8-40
NetWriteThreadWakeup (cache statistic)
 8-40
new
 region of temporary object memory 10-11
new users, adding 5-17, 5-30

- NewGemSizeBytes (cache statistic) 8-40
 - NewObjsCommitted (cache statistic) 8-40
 - NewSymbolRequests (cache statistic) 8-40
 - NewSymbolsCount (cache statistic) 8-40
 - NextSleepTime (cache statistic) 8-41
 - nil (predefined system object)
 - defined D-3
 - NotifyQueueSize (cache statistic) 8-41
 - NRS (network resource string) 3-15
 - GEMSTONE_NRS_ALL 3-16
 - syntax C-1
 - numberOfExtentRangesWithoutGC (System) 10-40
 - numberOfExtentsWithoutGC (System) 10-40
 - NumCacheWarmers (cache statistic) 8-41
 - NumInLdiQueue (cache statistic) 8-41
 - NumInMainInpQueue (cache statistic) 8-41
 - NumInNetReadWorkList (cache statistic) 8-41
 - NumInNetWriteQueue (cache statistic) 8-41
 - NumLiveSoftRefs (cache statistic) 8-41
 - NumNonNilSoftRefs (cache statistic) 8-41
 - NumRefsStubbedMarkSweep (cache statistic) 8-42
 - NumRefsStubbedScavenge (cache statistic) 8-42
 - NumSharedCounters (cache statistic) 8-42
 - NumSoftRefsCleared (cache statistic) 8-42
 - NumVotingSessions (cache statistic) 8-42
- O**
- object audit
 - and GemStone privilege 5-6
 - interpreting results of 8-16
 - repairing errors 8-16
 - statistics 8-16
 - object audits 8-12
 - Object Server, see *Stone repository monitor*
 - object table sweep
 - defined 10-7
 - objectAudit (Repository) 8-12, 8-14
 - progress count during 8-50
 - objectAuditNoReclaim (Repository) 8-13
 - ObjectMemoryGrowCount (cache statistic) 8-42
 - objects, large, identifying in the repository 10-44
 - ObjectsRead (cache statistic) 8-42
 - ObjectsRefreshed (cache statistic) 8-42
 - ObjectTablePageReads (cache statistic) 8-42
 - ObjsCommitted (cache statistic) 8-43
 - #objsMovedPerCommitThreshold 10-42
 - offline extent backup 9-33
 - restoring from 9-33
 - old
 - region of temporary object memory 10-11
 - OldestCrSession (cache statistic) 8-43
 - OldestCrSessNotInTrans (cache statistic) 8-43
 - oldestLogFileIdForRecovery (Repository) 7-8
 - oldestLogFileIdForRecovery (Repository) 7-9, 7-10
 - OldGenSizeBytes (cache statistic) 8-43
 - oldPassword:newPassword: (UserProfile) 5-31
 - and GemStone privilege 5-6
 - OmBytesFlushed (cache statistic) 8-43
 - online extent backup 9-3
 - restoring from 9-7
 - onlinebackup.sh
 - file in GemStone examples directory 9-6
 - OopNumberKHighWaterMark (cache statistic) 8-43
 - OopsReturnedByGemsCount (cache statistic) 8-43
 - /opt/gemstone directory 1-33, 2-8
 - /opt/gemstone/ used for GemStone files 1-33, 8-2
 - option value errors in configuration files A-11
 - OtherPageReads (cache statistic) 8-43
 - otherSessionNames (System)
 - and GemStone privilege 5-6
 - OtInternalPagesWrittenByGem (cache statistic) 8-43
 - OtInternalPagesWrittenByStone (cache

statistic) 8-43
 OtLeafPagesWrittenByGem (cache statistic)
 8-44
 OtLeafPagesWrittenByStone (cache statistic)
 8-44
 ownerAuthorization: (Segment) 5-25,
 5-40

P

page audit 8-10
 Page Manager
 defined 1-3
 log files 4-19, 8-3, 8-6
 page reclamation
 configuration parameters affecting 10-42
 page server process for GemStone 3-5
 AIO page server 1-44
 free frame page server 1-44
 tasks of 1-43
pageaudit command 8-10
 description B-9
 PageDisposesDeferred (cache statistic) 8-44
 PageIoCount (cache statistic) 8-44
 PageIoTime100SampleAvg (cache statistic)
 8-44
 PageIoTime10SampleAvg (cache statistic)
 8-44
 PageIoTimeOverallAvg (cache statistic) 8-44
 PageLocateCount (cache statistic) 8-44
 PageMgrPagesNotRemovedFromCachesCount
 (cache statistic) 8-44
 PageMgrPagesPendingRemovalRetryCount
 (cache statistic) 8-45
 PageMgrPagesReceivedFromStoneCount
 (cache statistic) 8-45
 PageMgrPagesRemovedFromCachesCount
 (cache statistic) 8-45
 PageMgrRemoveFromCachesCount (cache
 statistic) 8-45
 PageMgrRemoveFromCachesPageCount
 (cache statistic) 8-45
 PageMgrRemovePagesFromCachesPollCount
 (cache statistic) 8-45

PageMgrTimeWaitingForCachePgsvrs (cache
 statistic) 8-45
 PageReads (cache statistic) 8-45
 pageReads (System) 8-19
 PageReadsProcessingCommit (cache statistic)
 8-46
 PageReadsStoneCommit (cache statistic) 8-46
 PageReadsWaitingForCommit (cache
 statistic) 8-46
 pages
 defined 10-9
 reclaiming 10-35
 PageServersInCacheCount (cache statistic)
 8-46
 PagesNeedReclaimSize (cache statistic) 8-46
 and reclamation process 10-44
 PagesNotFoundInCacheCount (cache
 statistic) 8-46
 PagesNotRemovedFromCacheCount (cache
 statistic) 8-46
 PagesRemovedDirtyFromCacheCount (cache
 statistic) 8-46
 PagesRemovedFromCacheCount (cache
 statistic) 8-46
 PagesReturnedByGemsCount (cache statistic)
 8-47
 PagesWaitingForRemovalInStoneCount
 (cache statistic) 8-47
 pagesWithPercentFree: (Repository)
 6-13
 PageWaitQueueSize (cache statistic) 8-47
 PageWrites (cache statistic) 8-47
 pageWrites (System) 8-19
 parallel reclaim 10-10
 password
 defined 5-4
 modifying another user's 5-32
 modifying your own 5-20, 5-31
 see also *security*
 password:
 (UserProfile) 5-32
 and GemStone privilege 5-6
 passwords
 and GemStone privilege 5-6

- passwords, network 3-12
- passwords, shadowed 3-12
- pcmon.log 1-18
- perm
 - region of temporary object memory 10-11
- PermGenSizeBytes (cache statistic) 8-47
- permission, setting for executable configuration files A-7
- permissions
 - file 1-31
 - for Gem session processes 2-7
- PersistentPagesCommittedCount (cache statistic) 8-47
- PersistentPagesDisposed (cache statistic) 8-47
- pgsvr
 - GemStone page server process 3-5
- PgsvrCheckpointState (cache statistic) 8-47
- PgsvrWaitQueueSize (cache statistic) 8-48
- PinnedDataPagesCount (cache statistic) 8-48
- PinnedOtPagesCount (cache statistic) 8-48
- PinnedPagesCount (cache statistic) 8-48
- PinnedPrivatePagesCount (cache statistic) 8-48
- PinnedSharedCount (cache statistic) 8-48
- PinnedTotalCount (cache statistic) 8-48
- PlusInfinity (Float constant)
 - defined D-3
- PlusQuietNaN (Float constant)
 - defined D-3
- PlusSignalingNaN (Float constant)
 - defined D-3
- pom
 - region of temporary object memory 10-11
- PomGenScavCount (cache statistic) 8-48
- PomGenSizeBytes (cache statistic) 8-48
- POSIX AIO library (Linux only) B-15
- possible dead objects
 - defined 10-7
- PossibleDeadKobjs (cache statistic) 8-48
 - and reclamation process 10-44
- PostCheckpointPages (cache statistic) 8-49
- predefined system objects
 - AllGroups 5-25, 5-38, 5-40
 - AllUsers 5-30
 - NativeLanguage 5-9
- PreemptedBitmapPages (cache statistic) 8-49
- PreemptedCommitRecordPages (cache statistic) 8-49
- PreemptedDataPages (cache statistic) 8-49
- PreemptedObjectTablePages (cache statistic) 8-49
- PreemptedOtherPages (cache statistic) 8-49
- pre-growing repository extents 1-21, A-13
- presenting
 - error messages in the user's native language 5-9
- primitives, user-defined 2-12
- printing configuration options A-14
- privileged system functions, list of 5-6
- privileges
 - adding to a user's 5-5, 5-20, 5-33 and UserProfile 5-25, 5-40
 - assigning to a new user 5-20, 5-30
 - defined 5-4
 - deleting a user's 5-20, 5-34
 - examining a user's 5-5, 5-20, 5-32
 - GemBuilder Admin Tool and 5-20
 - redefining a user's 5-5, 5-34
 - required for system administration tasks 5-2
 - required to run epoch garbage collection 10-17
 - required to run markForCollection 10-26
 - required to run
 - markGcCandidatesFromFile : 10-34
 - required to run
 - sessionsReferencingOldes tCr 10-41
- privileges:
 - (UserProfile) 5-5, 5-32, 5-34
- ProcessId (cache statistic) 8-49

processing dead objects
and Admin GcGem 10-29
ProcessName (cache statistic) 8-49
.profile, captive account and 3-10
ProgressCount (cache statistic) 8-49
ProgressKobjs (cache statistic) 8-50
Published (dictionary)
and object sharing D-2
purging unneeded objects 10-2

Q

quickObjectAuditWithLevel
(Repository) 8-13

R

RAID devices 1-11
raw partitions
changing to and from 1-36
removing old contents B-10
setting up 1-34
use recommended 1-10
RcConflictCount (cache statistic) 8-50
RcRetryQueueSize (cache statistic) 8-51
read/write authorization 5-23, 5-39
and Segments 5-4
RebuildScavPagesForCommitCount (cache
statistic) 8-51
RecentActiveProcessCount (cache statistic)
8-51
Reclaim GcGem
and GemStone privilege 5-6
log files 4-19, 8-3, 8-6
Reclaim GcGems 10-8
and Admin GcGem 10-8
and multi-homed hosts 10-39
configuring 10-36, A-7
defined 1-3
reclaiming pages 10-35
running on remote nodes 10-39

ReclaimCount (cache statistic) 8-51
and reclamation process 10-44
#reclaimDeadEnabled 10-42
#reclaimDeadShadowPageThreshold
10-43
ReclaimedPagesCount (cache statistic) 8-51
and reclamation process 10-44
reclaimGcSessionCount (System) 10-40
reclaiming garbage in parallel 10-10
reclaiming pages 10-8
reclaiming system resources 10-2
Reclaim GcGems 10-8
#reclaimMinPages GcUser parameter
10-42
#reclaimSleepTime GcUser parameter
10-42
reclamation
tuning 10-41
RecoverTranlogBlockId (cache statistic) 8-51
RecoverTranlogFileId (cache statistic) 8-51
recovery
after file system repair 9-37
after full disk error 6-14
after NetLDI startup failure 4-9
after Stone startup failure 4-3
after unexpected shutdown 4-19
using GemStone full backup 9-14
using offline extent backup 9-34
redefining a user's privileges 5-5, 5-34
references to repository objects
deleting 10-46
searching for 10-45
remote logins
troubleshooting 3-30
RemoteCachesNeedServiceCount (cache
statistic) 8-51
RemoteSharedPageCacheCount (cache
statistic) 8-51
removedbf command
archiving transaction logs 7-8
description B-10
removeDictionary: (UserProfile) 5-35

- removeGroup: (UserProfile) 5-37
- removing
 - a user 5-19
 - a user from a group 5-23, 5-37
 - a user group 5-38
 - a user's privileges 5-20, 5-34
 - a UserProfile 5-19
- repairWithLimit: (Repository) 8-16
- RepBkupRestPagesWrittenByGem (cache statistic) 8-52
- RepBkupRestPagesWrittenByStone (cache statistic) 8-52
- repository
 - see also *extent files* and *transaction logs*
 - audit at object level 8-12
 - audit at page level 8-10
 - backups, see *backups*
 - bulk loading of 4-22
 - checkpoint frequency 1-42
 - disaster recovery B-16
 - disk full condition 6-14
 - free space in 6-3
 - growth of 10-1
 - identifying large objects in 10-44
 - marking garbage in 10-10
 - object references, deleting 10-46
 - object references, searching for 10-45
 - oldest log needed for recovery 7-9
 - page fragmentation 6-13
 - running multiple 1-46
 - running warm backup 1-48
 - shrinkage, when 10-8
 - shrinking to minimum size 6-10
 - shutting down 4-17
 - starting monitor 4-2
 - statistics from object audit 8-16
 - transaction logs, defined 1-4
 - updating views of extents 6-5
 - when free space appears in 10-35
- repository backup
 - offline extent 9-33
 - restoring from 9-33
 - online extent 9-3
 - restoring from 9-7
- repository below
 - freeSpaceThreshold (error message) 6-15
- repository, growth of 6-2
- Repository, single instance of D-3
- ReposSizeMB (cache statistic) 8-52
- resolving symbols, symbolList used in 5-8
- restore
 - and GemStone privilege 5-6
- restoreFromArchiveLogs (Repository) 9-21, 9-36
- restoreFromBackup: (Repository) 9-18
 - progress count during 8-50
- restoreFromBackups: (Repository) 9-24
- restoreFromBackupsNoShadows: (Repository) 9-24
- restoreFromCurrentLogs (Repository) 9-22, 9-37
- restoreFromLogDirectories: (Repository) 1-49
- restoreStatus (Repository) 9-18, 9-35
- restoreStatusOldestFileId (Repository) 7-10, 9-7, 9-32
- restoreToEndOfLog: (Repository) 9-21, 9-32, 9-36
- restoring
 - from an online extent backup 9-7, 9-33
 - from transaction log files 9-19
- restoring the GemStone repository
 - from a backup 9-14
 - performance tips 9-23
 - to a point in time 9-25
- resumeCheckpoints (System) 9-4, 9-5
 - and GemStone privilege 5-6
- resumeLogins (System)
 - and GemStone privilege 5-6
- Root40PagesWrittenByGem (cache statistic) 8-52

Root40PagesWrittenByStone (cache statistic) 8-52
RootPagesWrittenByGem (cache statistic) 8-52
RootPagesWrittenByStone (cache statistic) 8-52
RPC (remote procedure call) applications 3-4
 Gems A-7
rtErrSignalAbort 4-24
 and markForCollection 10-26
rtErrSignalAlmostOutOfMemory 10-17
rtErrSignalFinishTransaction 4-26
rtErrTranlogDirFull 7-14
RunQueueSize (cache statistic) 8-52

S

ScavengeCount (cache statistic) 8-52
ScavengeOverflows (cache statistic) 8-52
scheduling markForCollection 10-28
scratch directory, default A-14
search for
 executable configuration files A-5
 references to repository objects 10-45
 system-wide configuration file A-2
secure mode, NetLDI 3-9

security 5-42
 disabling inactive accounts 5-47
 finding disabled accounts 5-50
 last login by account 5-47
 limiting concurrent sessions by user 5-49
 login failures
 disabling further 5-49
 logging 5-49
 passwords
 aging 5-45
 clearing disallowed list of 5-45
 constraining choice of 5-42
 disallowing certain 5-44
 disallowing reuse of 5-44
 login limit under a 5-48
 warning of expiration 5-46
 when last changed 5-46
 see also *passwords*
segment
 changing a user's default 5-26, 5-41
 changing the authorization of a 5-25, 5-40, 5-41
 predefined D-4
 predefined segments D-4
 unit of authorization 5-4
 used in read/write authorization 5-23, 5-39
service name, GemStone 2-12
services.dat
 file in GemStone system directory 3-6
session statistics 8-21, 8-53
 _sessionCacheStatAt: (System) 8-22
 _sessionCacheStatsForProcessSlot: (System) 8-22
 _sessionCacheStatsForSessionId: (System) 8-22
SessionId (cache statistic) 8-52
SessionInBackup (internal parameter) A-38

- sessions
 - current session names 4-17
 - find who is logged in 4-16
 - finding process id of 4-17, 10-41
 - identifying current 10-41
 - Smalltalk methods that require explicit access privilege 5-6
- sessionSessionStatAt
 - put: (System) 8-22
- sessionsReferencingOldestCr
 - privilege required 10-41
- sessionsReferencingOldestCr (System) 10-41
- SessionStatn 8-53
- SessionWithGcLock (cache statistic) 8-53
- set command (Topaz) 5-28
- setArchiveLogDirectories: (Repository) 9-21, 9-33, 9-36
- setArchiveLogDirectories:tranlogP
 - refix: (Repository) 1-48
- setArchiveLogDirectory: (Repository) 9-33
- setArchiveLogDirectory:tranlogPre
 - fix: (Repository) 9-33
- setting
 - default size of Gem private page cache A-17
 - default size of stone page cache A-34
 - full transaction logging A-35
 - permission, executable configuration files A-7
- setuid bit
 - for Gem session processes 2-7
 - on executable files 1-31
- shadow object
 - contrasted with dead object 10-3–10-6
 - defined 10-3
- shadow objects
 - reclaiming pages from 10-8
- shadowed passwords 3-12
- ShadowedPagesCount (cache statistic) 8-53
- shared memory
 - access by Gems 2-7
 - utility to check system 1-18
- shared page cache
 - cleanup after **kill -9** 4-18
 - configuration 1-15
 - disconnect error 4-21
 - enabling 2-6
 - for remote Gem session processes 2-4, 2-6
 - maximum processes 1-18, A-22
 - monitor process 1-3, 1-15, 4-21
 - log file 8-3, 8-4
 - on remote machine 3-22
 - sessions on remote hosts and 3-6
 - size 1-16, A-23
 - spin lock attempts 1-41, A-23
 - stuck spin lock 4-21
 - timeout of remote A-34
- shared page cache monitor
 - log files 4-19
- shared symbol dictionaries 5-10
- shared system objects
 - in Globals dictionary D-2
- shared system objects, in Globals dictionary 5-9
- sharing objects
 - Published dictionary D-2
- shrinking the repository 6-10
- SHR_NUM_FREE_FRAME_SERVERS (configuration option) A-22
- SHR_NUM_FREE_FRAME_SERVERS (configuration option) 1-45
- SHR_PAGE_CACHE_LOCKED (configuration option) A-22
- SHR_PAGE_CACHE_NUM_PROCS (configuration option) 1-14, 2-6, 4-15, A-22
 - adjusting to number of users 1-17
- SHR_PAGE_CACHE_NUM_SHARED_COUNTERS (configuration option) A-22
- SHR_PAGE_CACHE_SIZE_KB (configuration option) 2-6, A-23

- shrpcmonitor 1-15
- SHR_SPIN_LOCK_COUNT (configuration option) A-23
- SHR_TARGET_FREE_FRAME_COUNT (configuration option) A-24
- shutdown (System)
 - and GemStone privilege 5-6
- shutdown message 4-20, 4-22
- sigAbort 1-42, 4-24
 - and markForCollection 10-26
 - configuration parameter controlling A-35
 - handler 4-25
- SigAbortsReceived (cache statistic) 8-53
- SigAbortsSent (cache statistic) 8-53
- SigLostOTRoot error A-28
- signal
 - sent on commit record backlog 4-24, 4-26
 - sent on lostOTRoot A-28
 - sent on low memory 10-17
 - sent on tranlogs full 7-14
- SleepDuringDisposeTempPageCount (cache statistic) 8-53
- #sleepTimeBetweenReclaim GcUser parameter 10-42
- SlotsCrashedCount (cache statistic) 8-53
- SlotsRecoveredCount (cache statistic) 8-53
- Smalltalk
 - compiler, and symbol resolution 5-8
 - kernel classes, and Globals dictionary 5-9
 - methods, and GemStone privileges 5-4
 - methods, calling C routines from 2-12
- Smalltalk full backups 9-8
- specifying size of extent files A-13
- SpinLockCount (cache statistic) 8-53
- SpinLockCount (internal parameter) A-23
- SpinLockFreeFrameSleepCount (cache statistic) 8-54
- SpinLockFreePceSleepCount (cache statistic) 8-54
- SpinLockHashTableSleepCount (cache statistic) 8-54
- SpinLockNewSymSleepCount (cache statistic) 8-54
- SpinLockOtherSleepCount (cache statistic) 8-54
- SpinLockPageFrameSleepCount (cache statistic) 8-54
- SpinLockSmcQSleepCount (cache statistic) 8-54
- standalone Gems A-7
- standby system, managing 9-41
- startAdminGcSession (System) 10-30
- startAdminGcSession(System) 8-14
- startAllGcSessions (System) 10-30, 10-37
- startAllReclaimGcSessions (System) 10-37
- startcachewarmer** command
 - description B-11
 - when to use 4-23
- startCheckpointAsync (System) 9-5
 - and GemStone privilege 5-6
- startCheckpointAsync(System) 7-12
- startCheckpointSync (System) 9-5
 - and GemStone privilege 5-6
- startCheckpointSync(System) 7-12
- starting GemStone 4-2
- startnetldi** command 3-4, 3-9
 - description B-13
- startNewLog (Repository) 7-11, 9-3
 - and GemStone privilege 5-6
- startReclaimGcSessions(System) 8-14
- startReclaimGemForExtentRange:to: (System) 10-38
- startReclaimGemForExtentRange:to: onHost: (System) 10-39
- startReclaimGemForExtentRange:to: onHost:stoneHost: (System) 10-39, 10-40
- startstone** command
 - description B-15
 - used in recovering from a backup 9-14
 - when restoring from backups 9-7
 - when transaction logs are missing 4-7

- statistics
 - filters for, in VSD F-13
 - global 8-22, 8-34
 - object audit 8-16
 - real-time monitoring F-5
 - session 8-21, 8-53
 - shared page cache 8-20
 - Smalltalk methods that require explicit privilege 5-6
- statmonitor
 - command line options F-14
 - purpose F-1
 - reference F-14
 - starting F-2
 - using F-1
 - viewing output files F-3
- STN_ADMIN_GC_SESSION_ENABLED (configuration option) 10-8, 10-29
- StnAdminGcSessionEnabled (internal parameter) A-24
- STN_ADMIN_GC_SESSION_ENABLED(configuration option) A-24
- StnAioCompletionFailures (cache statistic) 8-54
- StnAioFsyncFailures (cache statistic) 8-55
- StnAioSuspendEAGAIN (cache statistic) 8-55
- StnAioSuspendPrematureReturn (cache statistic) 8-55
- StnAioSyncWritesAfterCancel (cache statistic) 8-55
- StnAioWaitsForWork (cache statistic) 8-55
- StnAioWriteEAGAIN (cache statistic) 8-55
- StnAioWriteFailures (cache statistic) 8-55
- STN_ALLOCATE_HIGH_OOPS(configuration option) A-24
- STN_CHECKPOINT_INTERVAL (configuration option) 1-43, A-25
- StnCheckpointInterval (internal parameter) A-25
- STN_COMMIT_QUEUE_THRESHOLD (configuration option) A-25
- StnCommitQueueThreshold (internal parameter) A-25
- STN_COMMITS_ASYNC (configuration option) A-26
- STN_COMMIT_TOKEN_TIMEOUT (configuration option) A-25
- STN_CR_BACKLOG_THRESHOLD (configuration option) 4-24, A-26
- StnCrBacklogThreshold (internal parameter) A-26
- StnCurrentTranLogDirId (internal parameter) A-38
- StnCurrentTranLogNames (internal parameter) A-38
- STN_DISABLE_LOGIN_FAILURE_LIMIT (configuration option) 5-49, A-26
- StnDisableLoginFailureLimit (internal parameter) A-26
- STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT (configuration option) 5-49, A-26
- StnDisableLoginFailureTimeLimit (internal parameter) A-26
- STN_DISKFULL_TERMINATION_INTERVAL (configuration option) 6-15, A-27
- StnDiskFullTerminationInterval (internal parameter) A-27
- STN_EPOCH_GC_ENABLED (configuration option) 10-18, A-27
- StnEpochGcEnabled (internal parameter) A-27
- StnEpochGcThreshold (internal parameter) A-27
- STN_FREE_FRAME_CACHE_SIZE (configuration option) A-27
- STN_FREE_SPACE_THRESHOLD (configuration option) A-27
- STN_FREE_SPACE_THRESHOLD (configuration option) 6-14
- StnFreeSpaceThreshold (internal parameter) A-27
- STN_GEM_ABORT_TIMEOUT (configuration option) A-28
- StnGemAbortTimeout (internal parameter) A-28
- STN_GEM_LOSTOT_TIMEOUT (configuration option) 4-24, A-28
- StnGemLostOtTimeout (internal parameter) A-28

- STN_GEM_TIMEOUT (configuration option) A-29
- StnGemTimeout (internal parameter) A-29
- StnGetLocksCount (cache statistic) 8-55
- STN_HALT_ON_FATAL_ERR (configuration option) 1-30, A-29
- StnHaltOnFatalErr (internal parameter) A-29
- STN_HALT_ON_FATAL_ERROR (configuration option) 4-21
- StnLogGemErrors (internal parameter) A-38
- StnLoginsSuspended (internal parameter) A-38
- STN_LOG_LOGIN_FAILURE_LIMIT (configuration option) 5-49, A-30
- StnLogLoginFailureLimit (internal parameter) A-30
- STN_LOG_LOGIN_FAILURE_TIME_LIMIT (configuration option) 5-49, A-30
- StnLogLoginFailureTimeLimit (internal parameter) A-30
- StnLoopCount (cache statistic) 8-55
- StnLoopHibernateCount (cache statistic) 8-56
- StnLoopNoWorkThreshold (cache statistic) 8-56
- STN_LOOP_NO_WORK_THRESHOLD (configuration option) A-30
- StnLoopNoWorkThreshold (internal parameter) A-30
- StnLoopsNoWork (cache statistic) 8-56
- StnLoopsSinceSleep (cache statistic) 8-56
- StnLoopState (cache statistic) 8-56
- StnMainWaitsForFreeAio (cache statistic) 8-56
- STN_MAX_AIO_RATE (configuration option) A-31
- STN_MAX_AIO_REQUESTS (configuration option) A-31
- StnMaxReposSize (internal parameter) A-38
- STN_MAX_SESSIONS (configuration option) 1-18, 4-14, A-32
- StnMaxSessions (internal parameter) A-39
- STN_MAX_VOTING_SESSIONS (configuration option) A-32
- StnMaxVotingSessions (internal parameter) A-32
- StnMntMaxAioRate (internal parameter) A-31
- STN_NUM_GC_RECLAIM_SESSIONS (configuration option) 10-8, 10-36, A-32
- StnNumGcReclaimSessions (internal parameter) A-32
- STN_NUM_LOCAL_AIO_SERVERS effect when restoring backups 9-23
- STN_NUM_LOCAL_AIO_SERVERS (configuration option) 1-44, A-33
- STN_OBJ_LOCK_TIMEOUT (configuration option) A-33
- StnObjLockTimeout (internal parameter) A-33
- STN_PAGE_REMOVAL_THRESHOLD (configuration option) A-33
- StnPageRemovalThreshold (internal parameter) A-33
- STN_PRIVATE_PAGE_CACHE_KB (configuration option) A-34
- STN_REMOTE_CACHE_TIMEOUT (configuration option) A-34
- StnRemoteCacheTimeout (internal parameter) A-34
- StnShrPcTargetPercentDirty (internal parameter) A-34
- STN_SHR_TARGET_PERCENT_DIRTY (configuration option) A-34
- STN_SIGNAL_ABORT_CR_BACKLOG (configuration option) 1-42, A-35
- StnSunsetDate (internal parameter) A-39
- STN_TRAN_FULL_LOGGING (configuration option) 1-28, 7-3, 7-7, 7-12, A-35 and bulk loading of objects 4-23
- STN_TRAN_LOG_DEBUG_LEVEL (configuration option) A-36
- STN_TRAN_LOG_DIRECTORIES (configuration option) 7-1, 7-4, 7-7, 7-10, 7-13, A-36 and bulk loading of objects 4-23
- STN_TRAN_LOG_DIRECTORIES (configuration option) 1-47

STN_TRAN_LOG_LIMIT (configuration option) 1-43, 7-4, A-36

STN_TRANLOG_LIMIT (configuration option) and bulk loading of objects 4-23

StnTranLogLimit (internal parameter) A-36

StnTranLogOriginTime (internal parameter) A-39

STN_TRAN_LOG_PREFIX (configuration option) 7-1, A-36

STN_TRAN_LOG_SIZES (configuration option) 1-29, 1-30, 7-2, 7-10, 7-13, A-37

StnTranQToRunQThreshold (cache statistic) 8-56

STN_TRAN_Q_TO_RUN_Q_THRESHOLD (configuration option) A-37

StnTranQToRunQueueThreshold (internal parameter) A-37

Stone private page cache setting size of A-34 tuning 1-17

- Stone repository monitor
 - AIO page servers 1-44
 - checkpoint frequency 1-42
 - configuration
 - file 1-4
 - run time access to 1-38
 - sample files 1-5
 - configuring server 1-11
 - defined 1-3
 - disk usage 1-9
 - extents 1-19
 - on raw partitions 1-36
 - file descriptors for 1-13
 - file permissions for 1-30
 - Gem fatal errors, response to 1-30
 - identifying configuration file in use B-8
 - identifying sessions logged in 4-16
 - kernel parameters for 1-14
 - listing of 4-10
 - log files 4-19, 8-2, 8-3
 - memory for 1-12
 - private page cache 1-17
 - raw partitions 1-10
 - using 1-34
 - recovery 4-19
 - disk error 4-20
 - disk full condition 6-14
 - fatal error by Gem 4-21
 - shared page cache error 4-21
 - removing stale locks B-7
 - running multiple servers 1-46
 - running warm backup 1-48
 - security, see *security*
 - setuid bit and 1-31
 - shared page cache 1-15
 - diagnostics for 1-18
 - tuning of 1-41
 - shutting down 4-17
 - starting 4-2
 - troubleshooting 4-3
 - status of B-7
 - swap (paging) space for 1-13
 - swapping, excessive 1-42
 - transaction logs 1-27
 - on raw partitions 1-37
- StoneCommitState (cache statistic) 8-56
- stone.conf file A-8
- stoneConfigurationAt: (System) 1-38, 7-10
- stoneConfigurationReport (System) 1-38
- stopAdminGcSession (System) 10-30
- stopAllGcSessions (System) 10-31
- stopnetldi** command description B-17
- stopping GemStone 4-2
 - avoid **kill -9** 4-18
- stopSession: (System) 4-16, 8-14
 - delay for inactive sessions 4-16, 8-14
- stopstone** command
 - description B-18
 - shutting down repository 4-17
- stopUserSessions (System) 8-14
 - and GemStone privilege 5-6
 - delay for inactive sessions 8-14
- stuck spin lock error 4-21
- support, technical 1-vi
- suspendCheckpointsForMinutes: (System) 9-3, 9-5
- suspending checkpoints 9-3
- suspendLogins (System) 9-18
 - and GemStone privilege 5-6
- swap space
 - system needs for server 1-13
- swapping
 - reducing excessive 1-42, 2-12
- symbol dictionaries, shared among GemStone users 5-10
- Symbol Gem
 - defined 1-3
- Symbol List Browser tool 5-21
- symbol list, and UserProfiles 5-8
 - adding to 5-34, 5-35
 - removing from 5-35, 5-36
- symbol resolution 5-8

- SymbolCreationQueueSize (cache statistic) 8-56
 - SymbolGem
 - log files 4-19, 8-3, 8-7
 - SymbolUser
 - defined D-2
 - described 5-7
 - syntax
 - configuration files A-9
 - errors, in configuration files A-11
 - system
 - GemStone logs 8-2
 - objects, in Globals dictionary 5-9
 - shutdowns, diagnosing 4-19, 6-14
 - system clock 1-14
 - system control privilege, Smalltalk methods requiring 5-6
 - system objects
 - in Globals dictionary D-2
 - system.conf file
 - write access to 1-33
 - SystemRepository (predefined system object) defined D-3
 - SystemSegment (predefined system object) defined D-4
 - SystemUser
 - and AllUsers system object D-5
 - and SystemSegment D-4
 - defined D-1
 - described 5-7, 5-8
 - logging in as 5-2
 - system-wide configuration files
 - defaults A-12
 - defined A-1
 - search for A-2
- T**
- targeted marking 10-10
 - TargetFreeFrameCount (cache statistic) 8-57
 - TargetPercentDirty (cache statistic) 8-57
 - TCP/IP 3-5
 - technical support 1-vi
 - templates
 - format for visual statistics display F-16
 - using, in VSD F-13
 - _tempObjSpaceMax (System) 2-11, 10-15
 - TempObjSpacePercentUsed (cache statistic) 8-57
 - _tempObjSpacePercentUsed (System) 2-11, 10-15
 - _tempObjSpaceUsed (System) 2-11, 10-15
 - temporary object memory
 - and garbage collection 10-11
 - temporary object usage
 - examining 10-12
 - TempPagesDisposed (cache statistic) 8-57
 - time changes 1-14
 - TimeInCommitRecordDisposal (cache statistic) 8-57
 - TimeInFramesFromFindFree (cache statistic) 8-57
 - TimeInMarkSweep (cache statistic) 8-57
 - TimeInPgSvrNetReads (cache statistic) 8-57
 - TimeInPgSvrNetWrites (cache statistic) 8-58
 - TimeInScavenges (cache statistic) 8-58
 - TimeInStnGetLocks (cache statistic) 8-58
 - TimeInStonePageDisposal (cache statistic) 8-58
 - TimeInUpdateUnionsCommit (cache statistic) 8-58
 - TimeLastEpochGc (cache statistic) 8-58
 - TimePerformingCommit (cache statistic) 8-58
 - TimePerformingReadIo (cache statistic) 8-58
 - TimePerformingReadRequests (cache statistic) 8-58
 - TimeProcessingCommit (cache statistic) 8-59
 - TimerThreadWakeup (cache statistic) 8-59
 - TimeStoneCommit (cache statistic) 8-59
 - timeToRestoreTo: (Repository) 9-25
 - TimeWaitingForCommit (cache statistic) 8-59
 - TimeWaitingForIo (cache statistic) 8-59
 - TimeWaitingForStone (cache statistic) 8-59
 - TimeWaitingForSymbols (cache statistic) 8-59
 - TimeZone D-7–D-10

- Topaz
 - configuration files and A-8
- topaz**
 - command description B-19
- TotalAborts (cache statistic) 8-59
- TotalCommits (cache statistic) 8-59
- TotalGemFatalErrors (cache statistic) 8-60
- TotalLostOtsSent (cache statistic) 8-60
- TotalNewObjsCommitted (cache statistic) 8-60
- TotalSessionsCount (cache statistic) 8-60
- TotalSessionsStopped (cache statistic) 8-60
- TotalSessionsTerminated (cache statistic) 8-60
- TotalSigAbortsSent (cache statistic) 8-60
- TrackedSetSize (cache statistic) 8-60
- tranlog directories full (error message) 7-13
- TranlogFileId (cache statistic) 8-60
- TranlogRecordId (cache statistic) 8-60
- TranlogsFull (cache statistic) 8-60
- tranlogXXX.log (transaction log) 7-2
- transaction logging
 - comparison of full, partial 1-28, 7-3
 - enabling 1-27, 7-3, A-35
 - partial logging checkpoint threshold 1-27, 7-3, A-36
- transaction logs
 - adding online 7-10
 - archiving 7-8
 - backups for 7-8, 9-2
 - corrupted, recovering from 9-26
 - current log directory 7-10
 - current log file 7-10
 - current log size 7-10
 - disk full condition 7-13
 - disk space, managing 7-13
 - finding size and fileId of B-5
 - identifying a log file B-4
 - identifying checkpoints in B-5
 - log directories 7-2, A-36
 - log not found error 4-7
 - log origin time 7-10
 - log size limit 7-2, A-37
 - missing 9-27
 - moving to raw partition 1-37
 - oldest log needed for recovery 7-9
 - out of sequence 9-31
 - replaying on standby system 9-41
 - restarting stone without 4-7
 - restoring a subset of 9-30
- transaction mode, manual 8-1, 9-10
- transaction record backlog A-35
- TransactionLevel (cache statistic) 8-61
- transactionMode: (System) 4-25
- transactions
 - checkpoints
 - starting 7-12
 - restoring from log files 9-19
- transitive closure
 - defined 10-2
- troubleshooting
 - NetLDI startup 4-9
 - remote sessions 3-30
 - session login 4-14
 - Stone startup 4-3
- true (predefined system object)
 - defined D-3
- tuning reclamation 10-41

tz (TimeZone database) D-8
tzselect (TimeZone utility) D-9

U

UNIX

file system corruption 4-20
kernel configuration 1-19

UpdateUnionsCommitCount (cache statistic)
8-61

user accounts
administering 5-16

user actions, initializing 2-12

user groups
adding users to 5-23, 5-36
and AllGroups system object D-5
assigning a new user to 5-23, 5-30
creating 5-25, 5-40
defined 5-5
examining a user's memberships 5-23,
5-36
list all members of a 5-37
removing 5-38
removing a user from 5-37
removing a user from a 5-23
removing from a Segment's authorization
list 5-25, 5-41
used in Segment authorization 5-4

user passwords
and GemStone privilege 5-6

UserGlobals (user globals dictionary)
defined 5-9
initial contents of 5-9, D-2

userId: (UserProfile) 5-38

UserProfile
and AllUsers system object D-5
and GemStone privilege 5-6
creating a 5-17, 5-30
described 5-3
removing a 5-19

userProfileForSession: (System) 4-16
GemStone privilege and 5-6

users

access to network 3-13
adding 5-17, 5-30
current sessions 4-17
default segment 5-4
changing 5-26, 5-41
dictionaries 5-8
adding 5-21, 5-34
removing 5-35
disabling inactive accounts 5-47
environment variables E-1
finding disabled accounts 5-50
group membership 5-5, 5-23, 5-36
limiting concurrent sessions by same 5-49
listing all 5-17, 5-29
password 5-3
changing 5-20, 5-31
privilege 5-6
predefined users 5-7
privileges 5-4
changing 5-20, 5-33
reducing markForCollection's impact
on 10-28
removing 5-19, 5-38
security, see *security*
segment authorization 5-24, 5-39
sessions holding up reclamation 10-41
sharing objects among 5-10
symbol list 5-8
userId 5-3
changing 5-38
when last logged in 5-47
why account disabled 5-51
user-written C functions, calling from
Smalltalk 2-12
/usr/gemstone used for GemStone files
1-33

V

#verboseLogging GcUser parameter 10-43
verification of backups 9-14

visual statistics display
 configuration information F-16
 configuring F-16
 filtering statistics in F-13
 help F-5
 menu items F-5
 reference F-5
 starting F-3
 template format F-16
 using F-1
 using templates in F-13
 viewing statmonitor output F-3
 .vsdconfig F-16
 .vsdrc file F-16
 .vsdtemplates F-16

VoteNotDead (cache statistic) 8-61
VoteNotDeadKobjs (cache statistic) 8-61

voting
 defined 10-7

VSD
 help F-5
 menu items F-5

VSD: see visual statistics display

.vsdconfig F-16

visual statistics display
 .vsdconfig F-16

.vsdrc file F-16

.vsdtemplates F-16

W

waitForAllGcGemsToStartForUpToSeconds: (System) 10-30, 10-37

WaitingForSessionToVote (cache statistic)
 8-61

WaitsForOtherReader (cache statistic) 8-61

waitstone command
 example 3-15

waitstone command description B-20

weighted allocation of extents 1-23

WorkingSetSize (cache statistic) 8-61

worldAuthorization:
 (Segment) 5-25, 5-40

write authorization 5-23, 5-39

write set union sweep
 defined 10-7

writeFdcArrayToFile: (Repository) 10-33

Z

zdump (TimeZone utility) D-10

zic (TimeZone utility) D-10

zoneinfo (TimeZone database) D-8

