GemStone®

System Administration Guide for GemStone/S

for UNIX

Version 6.6

September 2011

VMware, Inc.
15220 NW Greenbrier Parkway
Suite 150
Beaverton, OR 97006

Preface

About This Manual

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

Prerequisites

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

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

How This Manual is Organized

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.
- Chapter 11, "Tuning Performance," describes how to diagnose and improve common performance bottlenecks.

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 E, “Environment Variables,” lists all environment variables used by GemStone, including those that are reserved.
- Appendix F, “Localization,” explains the syntax and semantics of the GemStone language-dependent file for messages and how you can create a similar file in another language, how to use Locale to handle non-US decimal points, and Extended Character Sets to allow sort and collation on Characters beyond the ASCII range.
- Appendix G, “statmonitor and VSD Reference,” describes how to use the performance-tuning tools statmonitor and VSD.
- Appendix H, “Object State Change Tracking,” describes how to analyze tranlogs to track the history of object modifications.

Terminology Conventions

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

Typographical Conventions

This document uses the following typographical conventions:

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

- Smalltalk methods, GemStone environment variables, operating system file names and paths, listings, and prompts are shown in `monospace` typeface.
- Interactive dialogue from GemStone is shown in an `underlined monospace` typeface.
- Lines you type are distinguished from system output by boldface type
- Place holders that are meant to be replaced with real values are shown in *italic* typeface.
- Literals are shown in **bold** typeface.
- Optional arguments and terms are enclosed in [square brackets].
- Braces { } mean 0 or more modifiers.
- Alternative arguments and terms are separated by a vertical bar (|).

Other GemStone Documentation

You may find it useful to look at documents that describe other GemStone system components:

- *Topaz Programming Environment* – describes Topaz, a scriptable command-line interface to GemStone Smalltalk. Topaz is most commonly used for performing repository maintenance operations.
- *Programming Guide for GemStone/S* – a programmer's guide to GemStone Smalltalk, GemStone's object-oriented programming language.
- *GemBuilder for Smalltalk Users's Guide* — describes GemBuilder for Smalltalk, a programming interface that provides a rich set of features for building and running client Smalltalk applications that interact transparently with GemStone Smalltalk.
- *GemBuilder for C* — describes GemBuilder for C, a set of C functions that provide a bridge between your application's C code and the application's database controlled by GemStone.

In addition, each release of GemStone/S includes *Release Notes*, describing changes in that release, and platform-specific *Installation Guides*, providing system requirements and installation and upgrade instructions.

A description of the behavior of each GemStone kernel class is available in the class comments in the GemStone Smalltalk repository. Method comments include a description of the behavior of methods.

Technical Support

GemStone Website

http://support.gemstone.com

GemStone's Technical Support website provides a variety of resources to help you use GemStone products:

- **Documentation** for released versions of all GemStone products, in PDF form.
- **Downloads** and **Patches**, including past and current versions of GemBuilder for Smalltalk.
- **Bugnotes**, identifying performance issues or error conditions you should be aware of.
- **TechTips**, providing information and instructions that are not otherwise included in the documentation.
- **Compatibility matrices**, listing supported platforms for GemStone product versions.

This material is updated regularly; we recommend checking this site on a regular basis.

Help Requests

You may need to contact Technical Support directly, if your questions are not answered in the documentation or by other material on the Technical Support site. Technical Support is available to customers with current support contracts.

Requests for technical support may be submitted online or by telephone. We recommend you use telephone contact only for serious requests that require immediate attention, such as a production system down. The support website is the preferred way to contact Technical Support.

Website: http://techsupport.gemstone.com

Email: techsupport@gemstone.com

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

When submitting a request, please include the following information:

- Your name, company name, and GemStone server license number.
- The versions of all related GemStone products, and of any other related products, such as client Smalltalk products.
- The operating system and version you are using.
- A description of the problem or request.
- Exact error message(s) received, if any, including log files if appropriate.

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

24x7 Emergency Technical Support

GemStone Technical Support 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 is available to help you succeed with GemStone products. Training for GemStone software is available at your location, and training courses are offered periodically at our offices in Beaverton, Oregon. Contact your GemStone account representative for more details or to obtain consulting services.

Contents

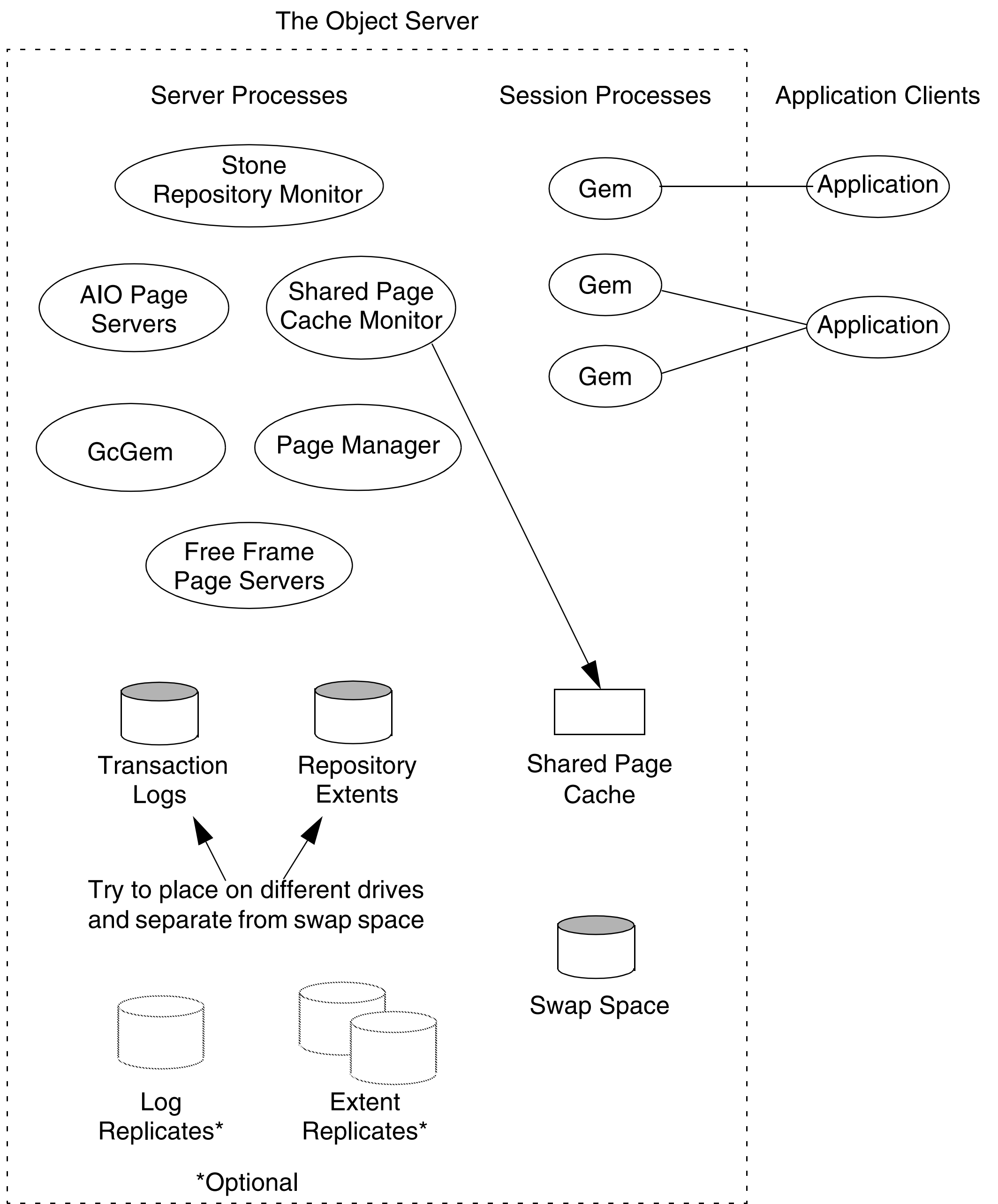
Chapter 1 Configuring the GemStone Server

Figure 1.1 shows the basic GemStone 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 *GcGem* is a Gem server process that is dedicated to performing the garbage collection tasks under supervision of 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, and optionally each extent can have a *replicated extent* or mirrored image to provide tolerance to disk failures.
- *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. Log files optionally can be replicated.

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.
- 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 four sample configurations that you can use as a starting point. (Sample configuration files are provided for three, as described later.) Although the configurations differ in a number of ways, the primary difference is in the size of application they accommodate.

All four 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. That configuration supports three concurrent user sessions and a repository of up to 100 MB.

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 `system.conf` *in the GemStone data directory.*

Small	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. The sample configuration file is `$GEMSTONE/examples/admin/small.conf`. Edit that file to specify the file name of your extent and the directory names for transaction logs.
Medium	uses raw disk partitions for possibly increased throughput. It accommodates more users and a larger repository. The sample configuration file is `$GEMSTONE/examples/admin/medium.conf`. Edit that file to show the raw partition name and size for your extent, and the partition names and sizes for transaction logs.
Large	uses multiple extents to accommodate a repository of several GB. The sample configuration file is `$GEMSTONE/examples/admin/large.conf`. Edit that file 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.
Very Large	accommodates up to 1000 users and a repository of 50 GB. Start with the sample configuration file for the Large configuration, but scale the configuration parameters by using the information in Tables 1.1 and 1.2. Each extent should be on a separate spindle.

To choose a sample configuration, select a column in the first part of Table 1.1 by matching the characteristics of your application to those shown. The lower part of that 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
Large and very 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 37.

Table 1.2 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.

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; and
- one spindle for transaction logs (on at least two raw partitions) and possibly user file systems if they are only lightly used for non-GemStone purposes.

The preceding configuration incorporates several guidelines to bear in mind when developing your own configuration. They are listed in order of importance:

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 okay to place multiple logs on the same spindle because only one log file is active at a time.

Table 1.1 Settings for Selected Configurations

Characteristic or Configuration Option	Server Configuration			
	Small	Medium	Large	Very Large
Application Characteristics				
Maximum number of user sessions	12	50	300	1000
Size of repository (GB)	0.100	1.5	16	50
System Requirements				
Typical number of CPUs	1–2	2	2–4	8 or more
Total real memory (MB)	128	512	2000	8000
Kernel shared memory (MB)	26	251	2004[c]	3006[d]
Number of disk drives	3[a]	3[a]	12[a]	24[a]
Configuration Settings				
STN_MAX_SESSIONS	40[b]	50	300	1000
STN_SIGNAL_ABORT_CR_BACKLOG	20	75	450	1500
STN_PRIVATE_PAGE_CACHE_KB	1600	3500	65000	65000
STN_NUM_LOCAL_AIO_SERVERS	1	1	5	10
STN_FREE_SPACE_THRESHOLD	10	50	100	100
SHR_PAGE_CACHE_SIZE_KB	25000	250000	1750000[c]	3000000[d]
Approximate Memory Usage				
Stone repository monitor (MB)	4	8	28	55
Each Gem session process (MB)	3.2	3.2	2.8	2.5

[a] These numbers do not allow for optional extent replication. If you replicate extents, each extent should be on an additional drive. If you replicate transactions logs, all replicate logs can share an additional drive.

[b] This setting is the initial setting supplied in system.conf.

[c] Make this setting the largest permitted by the operating system.

[d] Only Solaris supports a setting this large.

Table 1.2 Configuration Settings for Extents and Transaction Logs

Configuration Option	Server Configuration			
	Small	Medium	Large	Very Large
Extents				
DBF_EXTENT_NAMES	(1 file)	(1 raw partition)	(8 raw partitions)	(16 raw partitions)
DBF_EXTENT_SIZES	(unlimited)	1495[a]	(1995 each[a])	(3995 each[a])
DBF_PRE_GROW	False	True	True	True
DBF_ALLOCATION_MODE	(not used)	(not used)	10,10,...,10	10,10,...,10
DBF_REPLICATE_NAMES	(not recommended: OS-level mirroring is more efficient)			
Transaction Logs				
STN_TRAN_FULL_LOGGING	True	True	True	True
STN_TRAN_LOG_DIRECTORIES	(2 directories)	(5 raw partitions)	(5 raw partitions)	(8 partitions)
STN_TRAN_LOG_SIZES	10,10	199[a] each	499[a] each	499[a] each
STN_REPL_TRAN_LOG_DIRECTORIES	(not recommended: OS-level mirroring is more efficient)			
Stone Response to Gem Fatal Errors				
STN_HALT_ON_FATAL_ERR	False[b]	False[b]	False[b]	False[b]

[a] 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 Large configuration and 4 GB partitions in the Very Large configuration.

[b] For development and testing, a setting of True is recommended.
For deployed systems, a setting of False is recommended.

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 50.
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 63.

Developing a Replication Strategy

There are two needs to consider:

- Applications that cannot tolerate 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 (8 KB repository pages) and sequential access (up to 128 MB at a time). 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 be 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.

The DBF_REPLICATE_NAMES and STN_REPL_TRAN_LOG_DIRECTORIES configurations options listed in Table 1.2 provide an alternative means of increasing system tolerance to media failure. Changes to DBF_EXTENT_NAMES and DBF_REPLICATE_NAMES in the configuration file are all that is necessary before restarting GemStone. However, the cost of added I/O to maintain each replicate may be significant in some applications. The replicated extent should be on a different disk device (spindle) both for fault tolerance and to reduce I/O contention during ordinary operation.

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 and optional replicated extents.
5. Configure the transaction logs and optional replicated logs.
6. Set GemStone file permissions to allow necessary access while providing adequate security.

Gathering Application Information

You should have the following information at hand when you begin configuring GemStone because it is central to the sizing decisions you must make:

- 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), and
- the approximate size of your repository (it's also helpful, but not essential, to know the approximate number of objects in the repository).

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 object 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 the operating system and other software.

- The Stone and related processes need between 5.5 and 55 MB for the configurations shown in Table 1.1. 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 at least 4% to 10% of the repository size to provide adequate performance. In Table 1.1, the size ranges from tens of MB to three 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 2.5 MB of memory on the node where it runs (see the discussion of memory needs for session processes on page 82). 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

The total swap space on your system (sometimes called page space) in general should be at least twice the system RAM to provide reasonable flexibility. 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 depends on your operating system (examples are **swap**, **swapinfo**, **pstat**, and **lsvg**). The GemStone installation instructions for your platform contain an example.

Swap space should not be on a disk drive that contains any of the GemStone repository 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 the operating system 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 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.

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 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.

Shared Page Cache

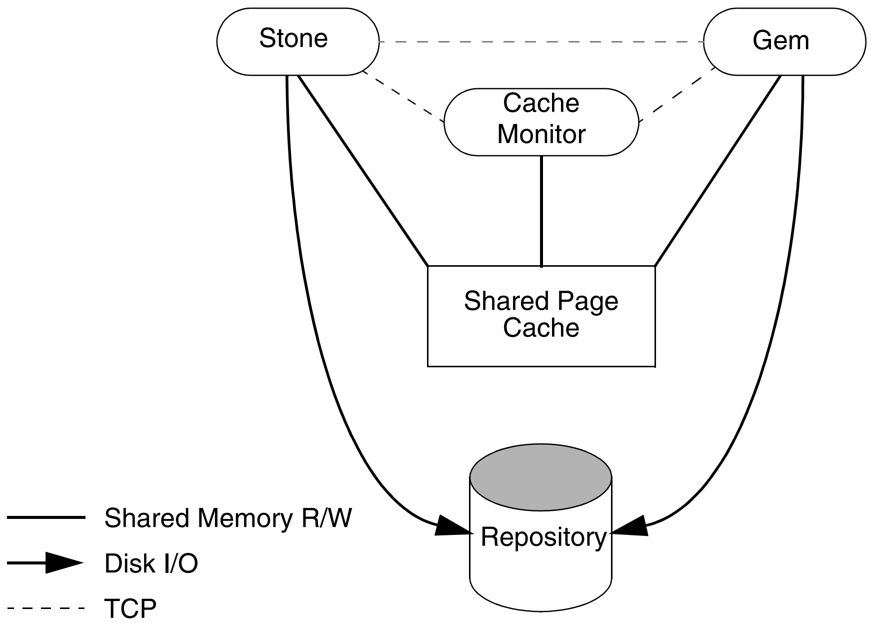
The GemStone shared page cache system consists of two components, the shared page cache itself and a monitor process (`shrpcmonitor`). Figure 1.2 shows the connections between these two and the main GemStone components when GemStone runs on a single node. There is no direct connection between the shared 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, `gemserver66@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 also 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 10 MB):

1. The number of objects in the repository

   We recommend that you allow room for one-third to one-half of the object table in the cache. Because each object uses four bytes in the table, use (2 bytes * *number-of-objects*) for this factor.

2. The size of the repository

We recommend that you keep between 3% and 8% of the repository in the cache:

add 8% of the first 100 MB of the repository to the previous total,
add 6% of the next 900 MB, and
add 3% of that portion greater than 1 GB.

3. The number of users and the size of their transactions

   We recommend adding 0.5 to 1 MB per user for most situations:

   add 1 MB per user for the first 10 users,
   add 0.5 MB per user for the next 40 users.

   For applications having more than 50 users, if the cache size computed thus far (in all three steps) is less than (0.8 MB * total_users), use the larger size.

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 "Monitoring Performance" on page 236, 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 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, which should be adjusted according to the number of sessions. The goal is to avoid having the Stone's private page cache overflow into the shared page cache, where it would waste valuable storage. The default size of 1 MB is sufficient for up to 5 sessions. Increase that setting by 1 MB for each additional 30 sessions.

Procedure

Follow these steps to configure the shared page cache and the Stone's private page cache:

Step 1. Set the SHR_PAGE_CACHE_SIZE_KB configuration option using Table 1.1 (on page 33) or your own estimate derived above (remember to convert to KB). Although we recommend this value as a starting point, smaller values can be used at the cost of increased disk activity. For instance, for the Medium configuration's 250 MB cache:

```
SHR_PAGE_CACHE_SIZE_KB = 250000;
```

Step 2. If the number of sessions will be greater than 40, increase the STN_MAX_SESSIONS configuration option accordingly. Make sure the SHR_PAGE_CACHE_NUM_PROCS option 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. If you expect more than five users, increase the Stone's private page cache by 40 to 64 KB per user. Add 40 KB per user for sessions that don't acquire locks, and 64 KB for sessions that acquire all three kinds. For instance, for 50 sessions and moderate use of locks, you might increase the default cache size of 1000 KB to 3500:

```
STN_PRIVATE_PAGE_CACHE_KB = 3500;
```

Step 4. 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, and

numProcs is either the SHR_PAGE_CACHE_NUM_PROCS setting or, if that is –1, (STN_MAX_SESSIONS + number_of_extents + SHR_NUM_FREE_FRAME_SERVERS + STN_NUM_LOCAL_AIO_SERVERS + 2).

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

```
% touch /tmp/shmem
% $GEMSTONE/install/shmem /tmp/shmem 100000 54
% 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 sessions 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. These 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, and 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 4 above). 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. Some operating systems report the descriptor limit in response to the C shell built-in command **`limit`**. You can also use the Bash built-in command **ulimit -a**, which produces a similar report.

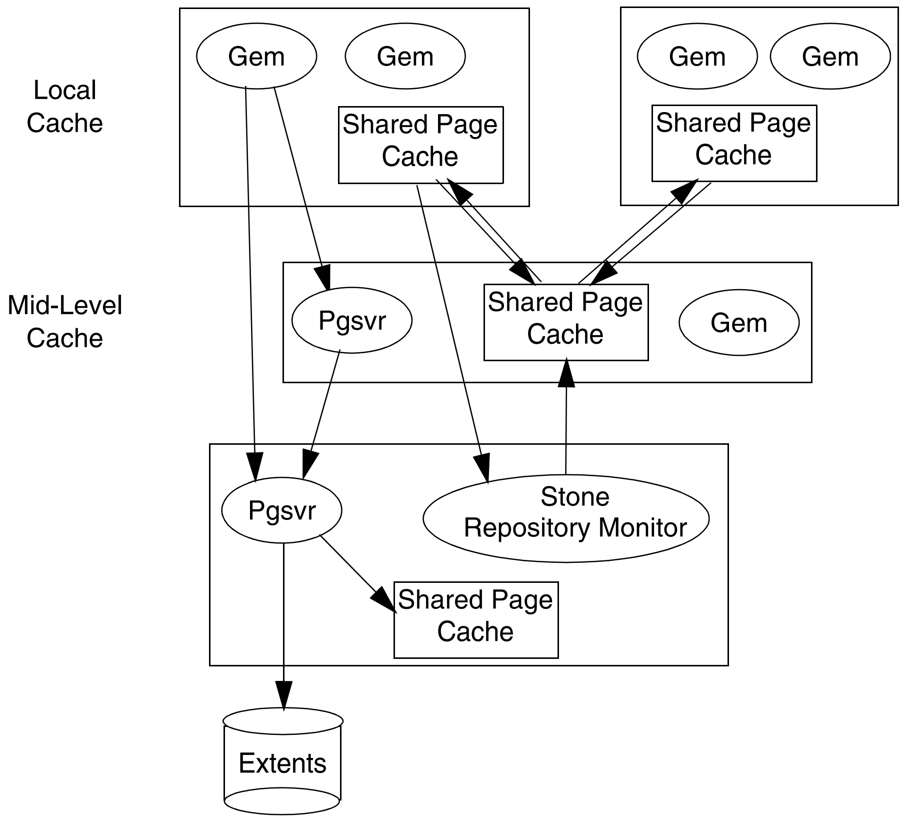
On operating systems that permit it, the shared page cache monitor attempts to raise the descriptor soft limit to the number required. In some cases, raising the limit may require super-user action to raise the hard limit or to reconfigure the kernel.

Using Mid-Level Caches

As described above, whenever a Gem session requests a Page Read, the request is forwarded directly to the page server on the Stone’s host. To reduce the amount of network traffic that would otherwise all go to the Stone's machine, you can set up *mid-level caches*.

As shown in Figure 1.3, if a Gem can't find a page in its local cache, it first looks in the mid-level cache. If the Gem can't find the page in the mid-level cache, it then forwards the request to the page server on the Stone’s host.

Figure 1.3 Using Mid-Level Caches



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

Mid-level caches may not be used in a system that uses replicate extents. You must reconfigure the stone to no longer use replicate extents before you can use any mid-level caches.

GemStone provides several methods in class System that let you connect to, and obtain information about, the mid-level caches on your system.

Connection Methods

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

`midLevelCacheConnect:` *hostName*

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

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

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

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

Reporting Methods

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

`remoteCachesReport`

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

`midLevelCachesReport`

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

To Configure the Repository Extents

Configuring the repository extents involves three primary considerations:

- providing sufficient disk space,
- minimizing I/O contention, and
- providing fault tolerance.

Estimating Extent Size

When you estimate the size of the repository, allow 10 to 20% for fragmentation. Also allow one-half MB of free space for each session that will be logged in simultaneously—if necessary, the extent will be expanded to provide this much head room. If the free space in extents falls below a level set by the STN_FREE_SPACE_THRESHOLD configuration option (the default is 1 MB), the Stone takes a number of steps to avoid shutting down. For information, see "How to Recover from Disk-Full Conditions" on page 202.

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
```

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:

- It's very important to keep extents on a spindle different from operating system swap space.

- Where possible, also 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/rdsk/c1t3d0s5, /dev/rdsk/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. 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 from the extent until that file system or raw partition is full or until 16 GB is reached.

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`. Use a comma to mark the position of an extent for which you do not want to specify a limit. For example, the following is for two extents of 500 MB each in raw partitions.

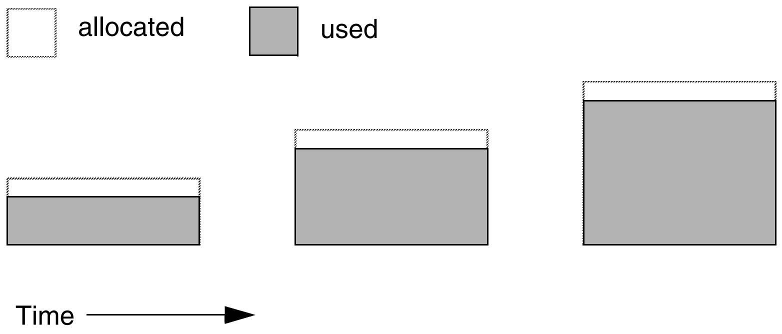
```
DBF_EXTENT_NAMES = /dev/rdsk/c1t3d0s5, /dev/rdsk/c2t2d0s6;
DBF_EXTENT_SIZES = 498, 498;
```

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

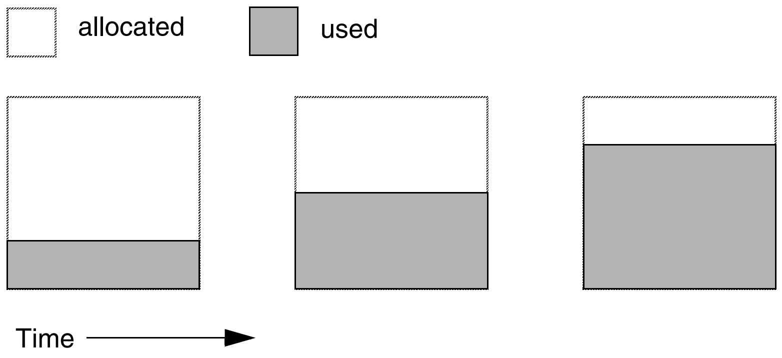
Figure 1.4 Growing an Extent on Demand



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 1.5 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 enables the operation, and DBF_EXTENT_SIZES 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. 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.

To pregrow extents, set both of the 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 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 1.6.) 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 (or zero), 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 avoiding the use of 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 of a repository's extents is to share the I/O load among the extents. For example, you can create three extents with equal weights, as shown in Figure 1.7.

Figure 1.6 Sequential Allocation

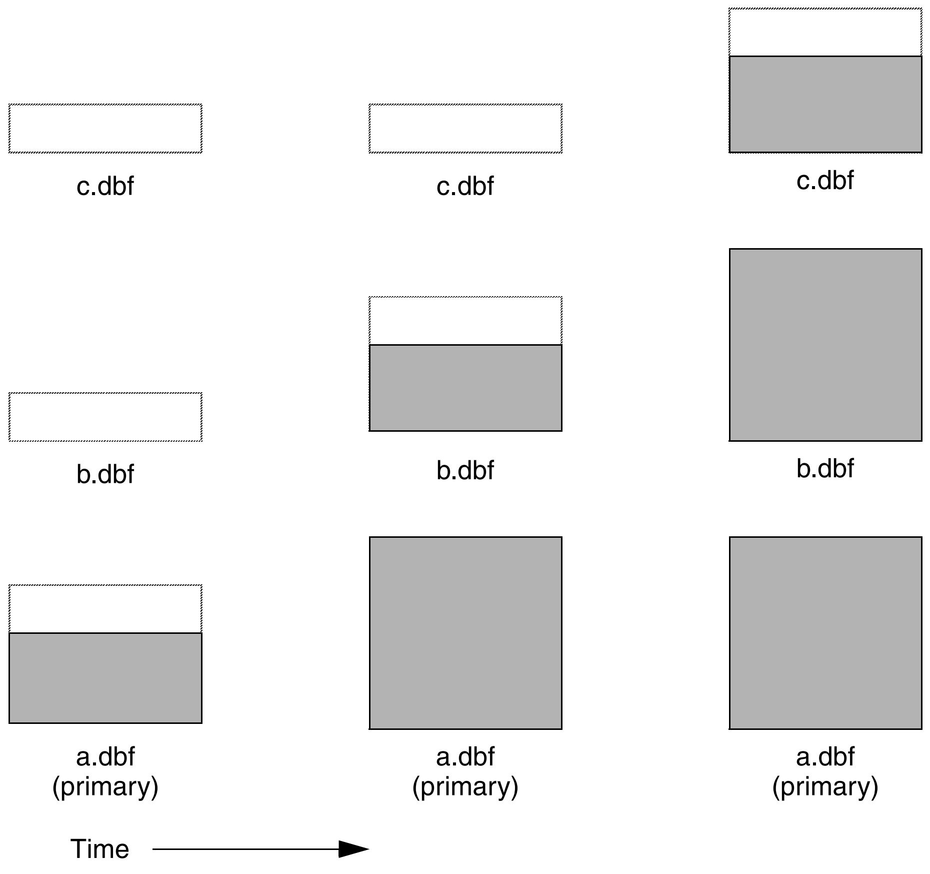
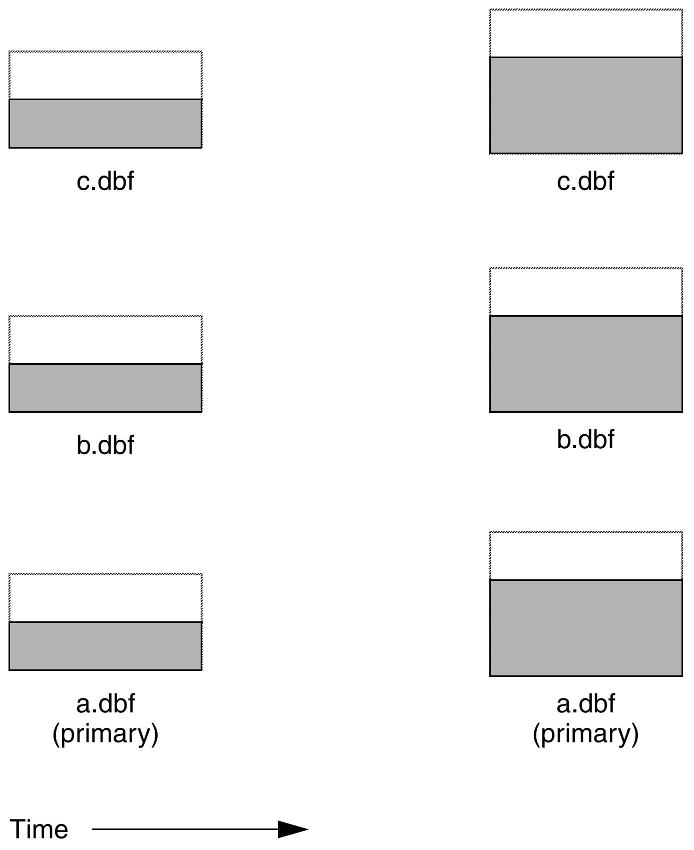


Figure 1.7 Equally Weighted Allocation

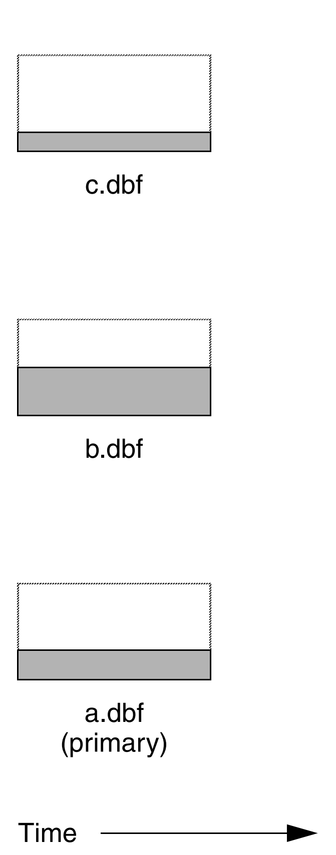
```
DBF_ALLOCATION_MODE = 10,10,10;
```



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 1.8 shows the result.

Figure 1.8 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.

NOTE
When you edit DBF_ALLOCATION_MODE, the number of weights you specify must match the number of files specified in DBF_EXTENT_NAMES and DBF_REPLICATE_NAMES.

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).

Replicating Extents

If you can afford the additional disk space and I/O overhead, a replicate (mirror copy) of your repository offers an excellent means of repository protection. It's best if the replicated extent is on a different disk drive and controller. You may not use mid-level caches if you use replicate extents.

A replicated extent replaces an extent at run time in the case of a read error. Should either a session process or the repository monitor encounter a read error on an extent, the replicated extent steps in and replaces that extent for the purposes of reading. (When the GemStone system encounters a write error on an extent, it terminates with an error message.)

The DBF_REPLICATE_NAMES configuration option determines which extents are replicated. Each entry must be one of three types:

- It may be the name of an existing replicated extent.
- It may be the name of a non-existent file; when the Stone repository monitor starts, it creates a new replicated extent by this name for the corresponding extent in `DBF_EXTENT_NAMES`.
- It may be an empty value, marked by a comma; the corresponding extent is not replicated, and any prior replicated extent is no longer updated (you should remove the outdated replicated extent).

An empty string (the default) means that no extents are replicated.

To replicate an extent, add the path of the replicated extent to DBF_REPLICATE_NAMES. This example uses raw partitions to replicate two extents:

```
DBF_REPLICATE_NAMES = /dev/rdsk/c3t2d,
/dev/rdsk/c3t2d0s30s3;
```

You can also create replicated extents at run time, although you must be the only user logged in. See "Repository>>createReplicateOf: named:" on page 192. You may also want to create replicates of transaction logs; see "To Add a Log and Replicate at Run Time" on page 215.

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, and
- providing fault tolerance, through both the choice of logging mode and the optional use of replicated logs.

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 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 218.

For general information about the logging mode and the administrative differences, see “Logging Modes” on page 208.

Estimating the Log Size

The disk space your application needs for transaction logs is highly individual because it depends on the logging mode you choose, your transaction characteristics, and 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 does this 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 like those for extents:

- It's very important to keep transaction logs on a different spindle than operating system swap space.
- Where possible, also 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.

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 limits.

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.

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.*

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/rdsk/c4d0s2,
/user3/tranlogs;
STN_TRAN_LOG_SIZES = 1998, 50;
```

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.

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

Replicating Logs

Because transaction logs are the primary element of real-time incremental backup under full transaction logging, you should consider maintaining replicated logs on another disk drive as protection against media failure. You can choose to replicate all of the logs or none of them.

When log replicates are in use, the Stone switches to new logs whenever either the primary (in STN_TRAN_LOG_DIRECTORIES) or the replicated log fills up or a write error occurs. If either of the new pair cannot be opened, processing continues using the other. On subsequent attempts to open a new log, the Stone again attempts to open both a primary and a replicated log.

During recovery from an unexpected shutdown, the Stone first tries to restore transactions by reading the primary transaction logs. If an error occurs in opening or reading one of the primary log files, the Stone attempts to read that information from the corresponding replicate log.

To create replicated logs, enter their location in the configuration file. Be sure you add as many replicate locations as there are primary log locations. You can list the same directory more than once, but a raw partition must have only one reference. You can mix raw partitions and files in the file system. For instance,

```
STN_REPL_TRAN_LOG_DIRECTORIES = /dev/rdsk/c4d0s3,
/user4/reptranlogs/;
```

You can add both a log and its replicate at run time if the existing logs are being replicated. See "To Add a Log and Replicate at Run Time" on page 215.

If you want to start replicating transaction logs for an existing repository, you must shut down the Stone repository monitor, edit the configuration file as explained above, and then restart the Stone.

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_ERR = 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, GcGem, 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 1.3 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 1.3 Recommended Resource and Process Permissions for the Server

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

[a] Ownership and permissions for the netldid executable depend on the authentication mode chosen and are discussed in Chapter 3.

If you are logged in as root when you run the GemStone installation program, it offers to set file protections in the manner described in Table 1.3. 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 84.

Alternative: Use Group Write Permission

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

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

Sites that run the linked version of GemBuilder may also prefer to use this protection so that file outs 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. By default, NetLDIs create log files in that 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/rdsk/c1t3d0s5`, `/dev/rsd2e`, or `/dev/vg03/rlvol1`. 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:

- 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, and
- it is not already being used for raw data.

When you select a partition, make sure 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.

Configure raw partitions as character devices, not as block devices.

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. **copydbf** will not overwrite an existing repository file. For instance,

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

Step 2. If you are not using an existing repository, make a local copy of the distribution extent. Using **copydbf** requires that you have write permission on the extent (or transaction log) that you are copying.

```
% cp $GEMSTONE/bin/extent0.dbf tempDirectory
% chmod +w tempDirectory/extent0.dbf
```

Step 3. 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 tempDirectory/extent0.dbf /dev/rsd2d
```

Step 4. 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` to further protect your repository. In particular, this executable file should not have the setuid (S) bit set.

Step 5. Edit the Stone's configuration file to show where the extent is located:

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

Step 6. 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. Similar changes can be made for transaction logs.

Extents

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 one or two 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. Set DBF_EXTENT_SIZES for this extent to be one or two 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.

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.

Transaction Logs

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 one or two 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. Make sure that STN_TRAN_LOG_SIZES for this location is one or two 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.

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 tranlog*nnn*.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. There are three access methods, which all provide similar server information (similar methods for accessing a session configuration are described on page 86):

`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.

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 lower case, 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 1.4 if you have the appropriate privileges. The options and parameters are described in Appendix A, "GemStone Configuration Options." 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
Configuration parameters should not be changed unless there is a clear reason for doing so, because incorrect settings can have serious adverse effects on GemStone performance. Check the entries in Appendix A for additional guidance about run-time changes to specific parameters.

Table 1.4 Server Configuration Parameters Changeable at Run Time

Configuration File Option	Internal Parameter
CONCURRENCY_MODE	#ConcurrencyMode
SHR_SPIN_LOCK_COUNT	#SpinLockCount[a]
STN_CHECKPOINT_INTERVAL	#StnCheckpointInterval[a]
STN_DISABLE_LOGIN_FAILURE_LIMIT	#StnDisableLoginFailureLimit
STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT	#StnDisableLoginFailureTimeLimit
STN_DISKFULL_TERMINATION_INTERVAL	#StnDiskFullTerminationInterval[a]
STN_FREE_SPACE_THRESHOLD	#StnFreeSpaceThreshold[a]
STN_GC_SESSION_ENABLED	#GcSessionEnabled[a]
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
STN_LOG_LOGIN_FAILURE_TIME_LIMIT	#StnLogLoginFailureTimeLimit
STN_REMOTE_CACHE_TIMEOUT	#StnRemoteCacheTimeout[a]
STN_SIGNAL_ABORT_CR_BACKLOG	#StnSignalAbortCrBacklog[a]
STN_TRAN_LOG_LIMIT	#StnTranLogLimit[a]
(none)	#StnLoginsSuspended

[a] These parameters can be changed only by SystemUser.

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: #GcSessionEnabled
%
-9
topaz 1> printit
System configurationAt:#GcSessionEnabled put: 0
%
0
```

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

1.5 How to Tune Server Performance

A number of configuration options are available for tuning the GemStone server to 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 tailor the shared page cache to the needs of your application. 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 according to the total number of objects in the repository and the number accessed at one time. Ideally, 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.

You should review the configuration recommendations given earlier ("Estimating the Size of the Shared Page Cache" on page 41) 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 "Monitoring Performance" on page 236, especially the statistic FreeFrameCount.

Matching Spin Lock Limit to Number of Processors

The SHR_SPIN_LOCK_COUNT configuration option specifies the number of times a process should attempt to obtain a lock in the shared page cache using the spin lock mechanism before resorting to setting a semaphore and sleeping. We recommend you leave SHR_SPIN_LOCK_COUNT set to –1 (the default setting), which causes GemStone to determine if 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 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 (given earlier) 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 a smaller proportion.

Excessive swapping also can be caused by the need to awaken (and swap in) sleeping sessions that are outside of a transaction. The Stone repository monitor takes this action (by sending a SignaledAbort message) when it runs out of space in the shared page cache 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. You can reduce this type of swapping activity by increasing the STN_SIGNAL_ABORT_CR_BACKLOG configuration option, which causes the Stone to keep more transactions in memory. 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 (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 one MB, assuming 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 and to any replicated extents. 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. A few Smalltalk methods, such as `System class>>checkpoint` and `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.

NOTE
If `System class>>checkpoint` *is called within a transaction that has not changed persistent data, then the underlying commit method is treated as an abort and the checkpoint doesn't occur, although the method still returns* `true`*.*

- 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. The default limit is 100 Kilobytes 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 CheckpointCount in the discussion of cache statistics on page 242.

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.

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, and
- 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 list 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 list 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 over four extents 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 421).

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 List 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 “GEM_FREE_FRAME_LIMIT” on page A-406.) 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 list page servers can sometimes help. On a single-CPU system, one extra free list 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 list page server process on its local node. Free list 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 list page servers will improve system performance. (For details about these and other statistics, see “Cache Statistics” on page 239.)

- 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 list page servers.
- If the FreeFrameCount is consistently lower than the FreeFrameLimit, a free list page server might help (though other factors enter into the question as well).

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

To Add Free List Page Servers

You can change the number of free list 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)

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, garbage collector session, and AIO page server.)
- 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 v6.6 NetLDI serves all Stones and Gem session processes on a given node.
- Multiple Stones can share a single installation directory, provided 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 32.

The following example shows the steps necessary to create a separate repository for application development (we'll identify it by the prefix *dev*). This repository will run in parallel with the initial repository that you installed by following the instructions in the *Installation Guide.*

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

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 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 that file instead of the default, which is `$GEMSTONE/data/system.conf`. For example:

(C shell)

```
$ setenv GEMSTONE_SYS_CONF /user2/devdata/system.conf
```

or (Bourne or Korn shell)

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

Step 4. Start the Stone for the development repository, giving it the name *devserver66.* GemStone will create log files with that server name as the prefix.

```
$ startstone devserver66
```

Step 5. Start linked Topaz, then set the GemStone name to *devserver66* and log in as DataCurator:

```
% topaz -l
topaz> set gemstone devserver66
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 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 / Warm Standby

Some customers may want to keep a duplicate of a production GemStone server running almost in parallel as a "warm" standby. The duplicate continually runs in restore mode, restoring each transaction log from the production server after the log is closed. If anything goes wrong with the primary production server, the warm standby can be brought into use very quickly.

This section tells how to set up the warm standby server and restore the logs. For general information about restoring backups and transaction logs, see "How to Restore a GemStone Repository" on page 281. This discussion assumes you are familiar with that procedure.

An important point to remember is that the transaction logs copied from the primary server, called the *archive logs* here, must be kept separate from the transaction logs created by the standby server. You can do that by using different log directories or different file name prefixes. If transaction logs are being replicated, the replicated logs also must be kept separate from those created by the warm standby.

To operate a warm standby, the server must be running in full logging mode.

Set up and run the warm standby

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

Step 2. Decide on a naming convention or location that you will use on the warm standby to keep the archive logs (the logs from the primary Stone) separate from those being created by the warm standby 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 warm standby server.

Step 3. Make a full backup of the primary server. (Instructions start on page 276.) 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. Restore the backup of the primary into the warm standby server, and leave the standby system running in restore mode. (Instructions for restoring backups start on page 281.).

Step 5. On the warm standby, tell the Stone where to find the archive logs by sending the following message:

```
Repository>>setArchiveLogDirectories:arrayOfDirectorySpecs
    tranlogPrefix:tranlogPrefixString
    replicateDirectories:arrayOfReplicateDirSpecs
    replicatePrefix:replicPrefixString
```

The arguments specify the directories (or raw partitions) to which the primary system's logs will be copied, and the log prefix they will have. 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 2:

```
topaz 1> run
SystemRepository setArchiveLogDirectories:
  #( '$GEMSTONE/data' )
tranlogPrefix: 'prodtranlog'
replicateDirectories: #()
replicatePrefix: '' .
%
```

Step 6. As each transaction log completes on the primary server, copy the log to the location that the standby is using for archive logs, and replay the transaction log using `Repository>>restoreFromArchiveLogs`. (More detailed instructions start on page 289.)

If the primary system's transaction logs are very large, they will be restored less frequently, but take a longer time to restore. If you need to get the standby system in operation quickly, this is a disadvantage. You may wish to limit the size of your transaction logs as described in "Choosing the Log Location and Size Limit" on page 58. When a transaction log grows to the specified limit, GemStone starts a new transaction log.

Alternately, you can force a new log on the primary server by sending `Repository>>startNewLog`. By running a script that starts a new log at regular intervals on your primary system, you can ensure that the warm standby is updated regularly regardless of the level of activity.

Continue repeating this step. You may find it necessary to shut down the standby from time to time. Ensure that you shut down the stone using stopstone. This does not affect the restore status, but you may need to update the archive log directories (see step 5).

Activate the warm standby in case of failure in the primary

Step 1. Replay the primary's latest transaction log on the standby system.

Step 2. Execute `Repository>>commitRestore` to terminate the restore process and enable logins.

Step 3. Client applications will have to reconnect to the standby system, which now becomes the primary system. Applications may have to perform their own failure recovery code as necessary, as well.

NOTE
Design your applications so that, after detecting a failure, they can determine which system is the new primary and reconnect correctly.

Step 4. Correct the problem on the failed system and restart it.

Depending on how much time has elapsed since the standby system became the primary system, either make a full backup of the new primary system and restore it on the system that failed, or replay the new primary system's transaction logs on the system that failed. Maintain that system in restore mode as the new standby.

Managing Page Reclamation in Warm Standbys

Indexing operations, migrations, `markForCollection`, and certain other operations can produce large numbers of pages with shadow objects. (For a detailed explanation of shadow objects and related concepts, see Chapter 10, “Managing Growth.”) Under ordinary system operation, these pages are reclaimed in the background without precluding foreground operations. However, when replaying transaction logs, the Stone can only reclaim these shadowed pages in the foreground, which can slow operation unacceptably.

Using the method `Repository>>restoreNoReclaimFromLog:` restores transaction logs more quickly, without performing reclaim; but these unreclaimed pages restored into the standby cannot be reclaimed in the background, and will cause excessive growth in the warm backup.

The Stone configuration parameter STN_RECOVERY_PAGE_RECLAIM_LIMIT provides control over the amount of reclaim that is done. This limits the maximum number of pages to reclaim for each transaction log record processed. The default value of this parameters is 2000. A transaction log is composed of many transaction log records, so using the default setting, 2000 pages are reclaimed many times within each transaction log. By setting this parameter to a much lower value, less reclaim is done, providing a compromise between standby growth and speed of replaying transaction logs.

For finer control, you may execute the method `Repository>>restoreReclaimPages` one or more times between replaying each transaction log. This method also uses the setting for STN_RECOVERY_PAGE_RECLAIM_LIMIT to determine how many shadow pages to reclaim each time it is executed.

By using the method `restoreNoReclaimFromLog:` instead of `restoreFromLog:`, and executing `restoreReclaimPages` in a loop between restoring transaction log, you can ensure that logs are restored as quickly as possible and that page reclamation be managed efficiently.

Chapter

2

Configuring Gem Session Processes

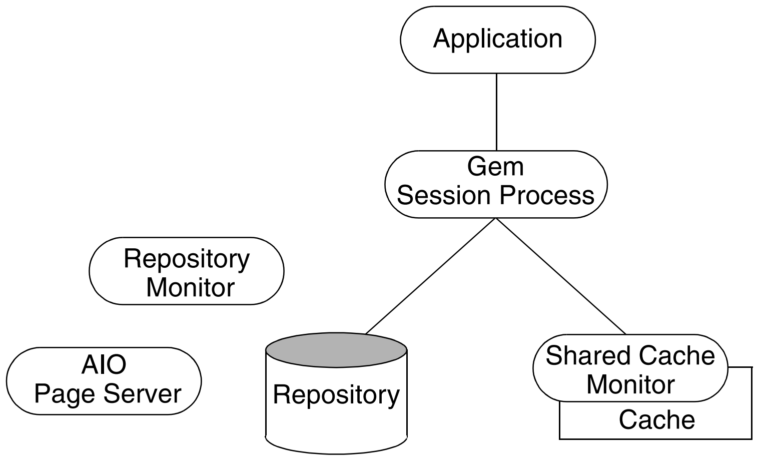
This chapter tells how to configure the GemStone session processes for your application. For additional information about running session processes on a node remote from the Stone repository monitor, refer also to Chapter 3.

2.1 Overview

A GemStone session involves six main components in a client-server relationship (Figure 2.1):

- 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, and
- 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 (currently 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. That appendix also describes each of the configuration options.

2.2 How to Configure Gem Session Processes

Configuring a Gem session process involves 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 (or disable) 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 first Gem session process on a node ordinarily requires about 4 MB of memory, of which 1.5 MB is for code that can be shared by other session processes. Each additional session process requires about 2.5 MB. The requirement is the same for Gems linked with an application. If you tune the cache size for Gems (page 88), add any increase to the amount given here. This memory is only for the session processes; for object server processes, see Chapter 1, "Configuring the GemStone Server."

There are additional memory needs on the server for Gem session process running on machines that are remote from the object server. For information, see "Estimating Memory Needs" on page 37.

Estimating Swap Space Needs

Swap (paging) space on machines remote from the Stones should follow the same general guidelines given on page 38 for servers. If you want to determine the additional swap space needed for GemStone session processes, use the memory requirements derived in the preceding section, including space for the number of sessions you expect. These 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.

You can use the GemStone **shmem** utility described on page 44 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.

❒ 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 60. Otherwise, the extents must be writable by a group to which the GemStone users belong (protection 660).

❒ 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 the 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 1.3 on page 61, the Stone process and shared page cache will belong to the owner you specify for those files (such as *gsadmin*).

What you need to do depends on these factors:

- whether the Gem session process is linked with the application or RPC,
- whether the Gem session process runs on the server or on a node remote from the Stone, and

- whether the server uses setuid bit and protection mode 600 for the extents (as recommended on page 60) or uses the alternative of group write permission.

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 61. 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_DEFAULT_NRS can be used to specify an alternative location; see "To Set a Default NRS" on page 107.

`/opt/gemstone`	All users should have read/write/execute access to the directories `/opt/gemstone/log` and `/opt/gemstone/locks` on each host. (For compatibility with previous releases, these directories can be in `/usr/gemstone`.) NetLDIs create their own log files in that log directory. GemStone processes that have a name for each instance (currently the Stone repository monitor, the shared page cache monitor, and the NetLDI) create lock files in the locks directory.

2.3 How to Access the Configuration at Run Time

GemStone provides 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. Four of these parameters are internal only; that is, they do not have counterparts in the configuration file. 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
Configuration parameters should not be changed unless there is a clear reason for doing so, because incorrect settings can have serious adverse effects on GemStone performance.

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_NATIVE_CODE_MAX	#GemNativeCodeMax
GEM_NATIVE_CODE_THRESHOLD	#GemNativeCodeThreshold
GEM_PGSVR_COMPRESS_PAGE_TRANSFERS	#GemPgsvrCompressPageTransfers
GEM_TEMPOBJ_CACHE_SIZE	#GemTempObjCacheSize
(none)	#NotConnectedDelta
(none)	#NotConnectedThreshold

The following example first obtains the value of the key #GemTempObjCacheSize. Since that is a negative SmallInteger, the parameter is one that can be changed at run time (this one can be raised, but attempts to lower it generate an error):

```
topaz 1> run
ConfigurationParameterDict at: #GemTempObjCacheSize
%
-28
topaz 1> run
System configurationAt:#GemTempObjCacheSize put: 1000
%
1000
```

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

Increase GEM_TEMPOBJ_CACHE_SIZE for applications that create a large number of temporary objects. Examples are applications making heavy use of the reduced conflict classes or sessions performing a bulk load. It is important to provide sufficient temporary object space because overflows are written into the session process's private page cache, which is discussed just ahead. Such overflows are costly because they force the use of page-size units for allocating and reclaiming storage space.

You will probably need to experiment somewhat before you determine the optimum size of the temporary object space for an application. In general, keep the size somewhere between 400 KB and 3 MB. Large applications typically require a temporary object space of 1 to 1.5 MB. You may find it helpful to examine the cache statistics NotConnectedObjsSetSize and MakeRoomInOldSpaceCount; see "Monitoring Performance" on page 236.

As shown in Table 2.1, the temporary object space can be increased at run time by setting the parameter #GemTempObjCacheSize, although this change should only be made immediately after logging in.

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

To Tune the Private Page Cache

Increase the GEM_PRIVATE_PAGE_CACHE_KB setting for Gems that modify a large number of objects or perform repository maintenance operations. This configuration option controls a *private page cache* that each session process uses to store objects created by an application. While the temporary object space (above) reads and writes memory on a per-object basis, the private page cache reads or writes a page (8 KB) at a time. When you commit objects created by an application, they move first from temporary object space to the session's private page cache.

If temporary object space overflows, objects are written into the session's private page cache. Although some temporary objects that overflow can be reclaimed in memory as part of the Gem's notConnectedSet (next), others may be reclaimed only after the page is written to the disk. As a result, overflowing temporary object space can use storage inefficiently, as well as waste the time required to reclaim that storage.

If you need to increase either temporary object space or the session's private page cache, increase the temporary object space first. Because it deals with objects one at a time instead of in page-size increments, and because the object's storage can be reclaimed more efficiently, temporary object space can deal more effectively with temporary space requirements.

The sum of the temporary object space and the private page cache needs to be larger than the default values for these caches only if a typical transaction commits more data than their combined size.

To Limit the Session I/O Rate

It may be desirable in some cases to limit the I/O rate of a particular Gem session process to reduce its interference with other GemStone activity. Two examples are administrative sessions doing `Repository>>markForCollection` or `fullBackupTo:`, which may involve considerable disk I/O over an extended period.

The I/O rate can be limited either by changing the configuration file read by a particular session process when it starts or by changing the corresponding internal parameter at run time. (You can cause a session process to read a particular configuration file by setting the `GEMSTONE_EXE_CONF` environmental variable; see "Search for an Executable Configuration File" on page 396.)

The following example sets an I/O limit of 10 per second in the configuration file.

```
GEM_IO_LIMIT = 10;
```

The default limit of 5000 I/Os per second essentially makes the rate limited only by performance of the underlying file system and disk partitions.

To change the limit at run time, use the internal parameter `#GemIOLimit` for the current session. For general information about such changes, see "To Change Settings at Run Time" on page 87.

The UserGlobals for GcUser has an association with the same key, `#GemIOLimit`. Its value is monitored and used as the GcGem I/O limit if the value is a SmallInteger greater than 1. Any user with privilege to write the GcUser's segment can update this parameter to control the GcGem process.

Changing the I/O Limit During a Long Operation

Privileges required: SystemControl.

NOTE
The following procedure is intended for experienced GemStone users and should not be necessary under ordinary circumstances.

Changes to `#GemIOLimit` may go unnoticed while the Gem is executing a long-running operation, such as `markForCollection`, `fullBackupTo:`, and `objectAudit`. To change the I/O limit during such operations, you must log in to a different session and communicate with the Gem by way of its shared page cache slot. Because the sessions must communicate through a single shared page cache, their Gems must run on the same node.

Step 1. Determine the shared page cache slot being used by the Gem for which you want to change the limit. You can find the slot number by finding the other Gem's operating system processId and then invoking `System class>>cacheStatistics:` *aSlot* for successive slots until you obtain a match in element 2, which is the processId for that slot.

Step 2. Send the message `changeCacheSlotIoLimit:`*aSlot* `to:` *aValue* to System. For example, to change the I/O limit to 100 per second for the Gem attached to cache slot 8:

```
topaz 1> run
System changeCacheSlotIoLimit: 8 to: 100
%
```

For information about the cache statistic itself, see "MilliSecPerIoSample (Stone)" on page 255.

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. The Stone takes this action (by sending a SignaledAbort message) when it runs out of space in the shared page cache 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 69.

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 scripts `gemnetobject` and `gemnetobjcsh` from `$GEMSTONE/sys` to your working directory. Those shell scripts are used to start Gem session processes under the Bourne or the Korn shell and the C shell, respectively. You will modify these scripts 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. Repeat the previous step for `gemnetobjcsh`.

Step 4. Rename your modified copies of the shell scripts `gemnetobject` and `gemnetobjcsh` so that they have distinct file names. For example:

```
% mv gemnetobject MyGemnetObject
% mv gemnetobjcsh MyGemnetObjcsh
```

Step 5. Copy the new shell scripts to `$GEMSTONE/sys`. Make sure that all GemStone users have read and execute (**r-x**) permission for those scripts. For example:

```
-r-xr-xr-x 1 root 912 Feb 24 20:22 MyGemnetObject
-r-xr-xr-x 1 root 863 Feb 24 20:22 MyGemnetObjcsh
```

If necessary, change the permissions:

```
% chmod 555 MyGemnetObject
% chmod 555 MyGemnetObjcsh
```

Step 6. Add entries for the new shell scripts to the services database, `$GEMSTONE/sys/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
MyGemnetObjcsh $GEMSTONE/sys/MyGemnetObjcsh
```

Step 7. Copy the new Gem executable to the GemStone system directory. For example:

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

Step 8. 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.

Because `gemnetobject` executes the user's `.profile`, some users of the Korn shell may encounter errors if their `.profile` contains commands that are not POSIX compliant. Such users should place the non-compliant ksh commands within a conditional like that shown on page 140.

Chapter
3
Connecting Distributed Systems

This chapter tells how to set up GemStone in a distributed environment:

- *Overview* (page 94) — An introduction to the GemStone processes and network objects that facilitate distributed GemStone systems.
- *How to Arrange Network Security* (page 102) — Three ways to provide access to GemStone processes on other nodes.
- *How to Use Network Resource Strings* (page 107) — How to specify where distributed GemStone resources are located.
- *How to Set Up a Remote Session (*page 109*)* — 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. 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.

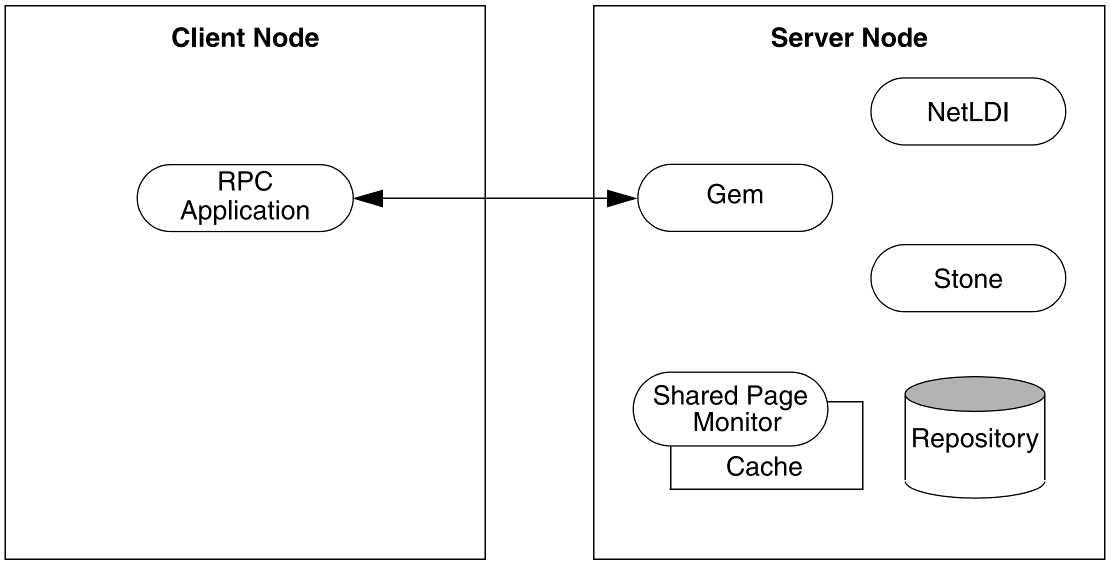
Figure 3.1 shows two typical distributed configurations in which an application on a remote (client) node is logged in to a repository and Stone repository monitor running on a server node.

In the configuration shown at the top of 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 115.

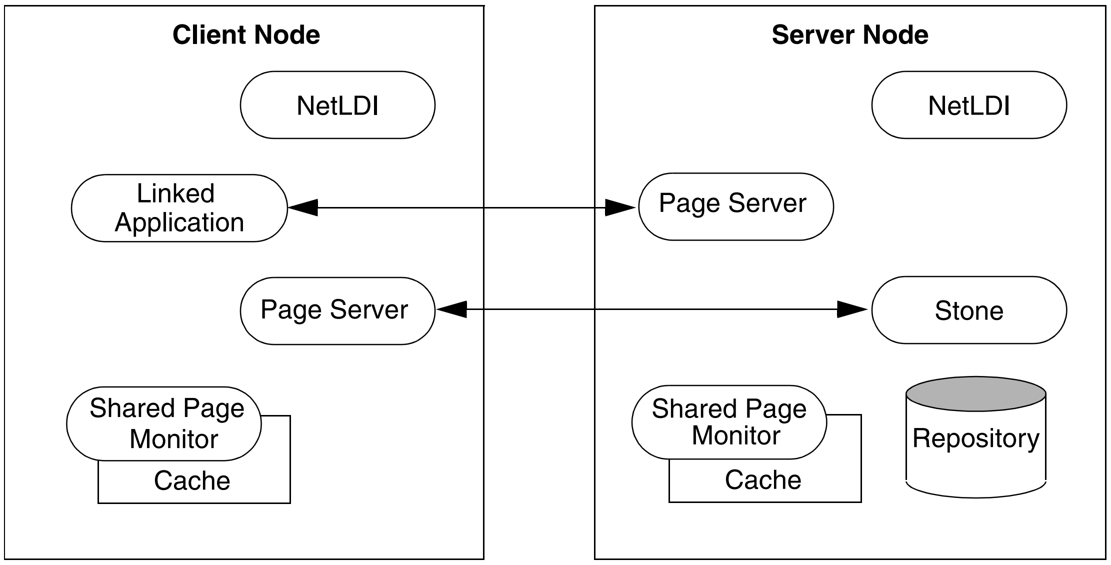
At the bottom of Figure 3.1, the application and the Gem are linked in a single process that runs on the client node. This configuration avoids the overhead of RPC calls, but in some applications it may increase network traffic substantially if large objects must be brought across the network. Ordinarily, the Stone repository monitor starts a shared page cache on the client 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 112.

Figure 3.1 Typical Distributed Configurations

A. Gem Session Process Runs on Server Node



B. Gem Session Process Runs on Client 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 the RPC applications run.)

You start a NetLDI directly by invoking the **startnetldi** command. 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 60. 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`. 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.

Captive Account Mode

GemStone administrators can force child processes to belong to a single, designated account by starting a NetLDI in *captive account mode* (**startnetldi -a***name*). All processes created by the NetLDI will belong to account *name,* which provides additional security. 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 106.

The captive account can be an ordinary user account or one created for that purpose, such as a GemStone administrative account. Child processes will read the shell initialization file (`.cshrc` or `.profile`) from the captive account, and 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.

NetLDI Names

The default name of the NetLDI process is `netldi30`. 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,

  (C shell)
  ```
  % setenv GEMSTONE_NRS_ALL \#netldi:netLdiName
  ```
  or (Bourne or Korn shell)
  ```
  $ GEMSTONE_NRS_ALL=#netldi:netLdiName
  $ export GEMSTONE_NRS_ALL
  ```

For more information about GEMSTONE_NRS_ALL, see "To Set a Default NRS" on page 107.

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 71.

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 object* 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, page manager, and the garbage collection (GcGem) session.

The network object most visible to users is the Gem session process requested by an RPC application. This object can be `gemnetobject`, `gemnetobjcsh`, 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/sys/services.dat.` There are two entries for the standard Gem session process:

```
gemnetobject            $GEMSTONE/sys/gemnetobject
gemnetobjcsh            $GEMSTONE/sys/gemnetobjcsh
```

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/sys/gemnetobject`. Similarly, an object name can be entered while setting up a GemBuilder session (as Name of Gem Service) or other application. Application programmers provide the name as a parameter to `GciSetNet().`

Notice that GemStone provides two services with similar names, `gemnetobject` and `gemnetobjcsh.` The former is a Bourne shell script, and the latter is a C shell script. Each script tries to read the user’s shell initialization file (`.profile` or `.cshrc`). The initialization file makes a difference principally with methods such as `System class>>performOnServer:`, which can take environment variables as arguments.

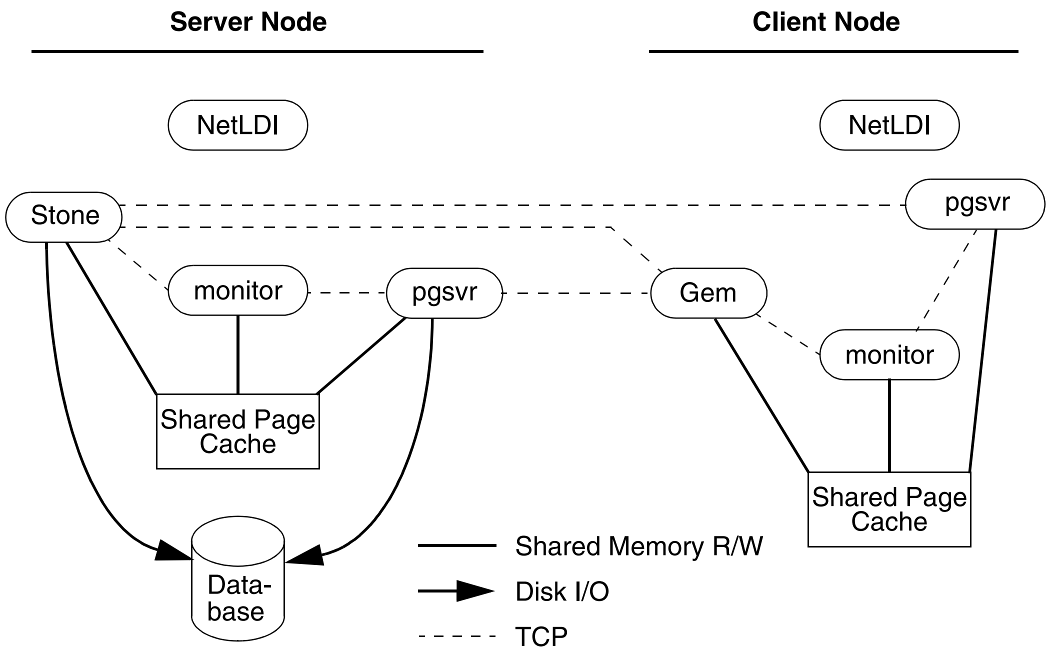
If your application uses a custom Gem executable, you can edit `services.dat` to include the appropriate mapping. For the procedure, see “How to Install a Custom Gem” on page 91.

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 path name.

Shared Page Cache in Distributed Systems

Gem session processes on a client node create a remote shared page cache, as shown in Figure 3.2. When the remote session logs in to the repository, the Stone repository monitor uses a NetLDI and page server (`pgsvrmain`) on the client 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 client 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.

Figure 3.2 Shared Page Cache with Remote Gem



Distributed Systems over a WAN

When Gem processes are distributed over a Wide Area Network (WAN), the distance between the Gem and Stone can slow down communication and negatively impact performance.

If a distributed system includes Gems that are running on more than one machine far from the stone, all the Gems that are physically close to each other relative to the WAN topology should use a mid-level cache close to those gems. To set up mid-level caches, see "Using Mid-Level Caches" on page 45.

The Gem configuration parameter GEM_PGSVR_COMPRESS_PAGE_TRANSFERS configures compression of page transfers between a Gem and its pgsvr process on the Stone's machine, and between the Gem's pgsvr on a mid-level cache and the Gem's pgsvr on the Stone's machine. Page transfers between a gem and a pgsvr on a mid-level cache are not compressed, since it is assumed that mid-level cache is on a node close to the gem process.

To further reduce the number of round trips to the stone, a Gem can be configured to get more free pages from Stone each time it needs free pages, using the configuration parameter GEM_FREE_PAGEIDS_CACHE.

Disrupted Communications

Several incidents can disrupt communications between the GemStone server and its clients 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.

The commands are:

Solaris (Sun):

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

AIX:

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

Linux:

`/sbin/sysctl -w net.ipv4.tcp_keepalive_time = <value>` (or add this entry to the `/etc/sysctl.conf` file)

3.2 How to Arrange Network Security

The system administrator can set the GemStone authentication requirement to one of three levels:

- 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,
  - starting a Stone repository monitor when an extent or transaction log resides on a node remote from the Stone,
  - creating or restoring a GemStone backup using a device on a node remote from the Gem performing the operation, and
  - 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.
- **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.

- **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 (page 96). Table 3.1 shows how guest mode and captive account mode affect NetLDI operation.

Table 3.1 Effect of NetLDI Guest Mode and Captive Account Options

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
-a*name*	Yes, for RPC Gem or copydbf between nodes	Account *name* (must start the NetLDI)	Ordinary user (*name*)	Owner of NetLDI
			Root	Any user
-g	No	Client's account	Ordinary user	Owner of NetLDI
			Root—not allowed	
-a*name* **-g**	No	Account *name* (must start the NetLDI)	Ordinary user (*name)*	Any user
			Root—not allowed	

For a complete list of the options to **startnetldi**, see the command description on page 443.

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

- password authentication (the default) with the NetLDI running as root, and
- guest mode combined with a captive account.

Examples later in this chapter include procedures for specific configurations (see "Configuration Examples" on page 111).

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,
- enter the login name and password through the application's user interface, such as the HostUserName and HostPassword parameters in Topaz, or
- use the NRS authorization modifier **#auth:***loginName@password* as part of a process name or file name.

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,
- starting a Stone repository monitor for which an extent resides on a node remote from Stone,
- creating or restoring a GemStone backup using a device on a node remote from the Gem performing the operation, or

- 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). The `.netrc` file should contain one line like the following for each node:

```
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 provides similar fields in its login dialog: **Host username** and **Host password**.

Using an NRS #auth Modifier

A final way is to provide the name and password 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 96) 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 described on page 60 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/sys/services.dat`.) If you prefer, such ad hoc services can be prohibited 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). Instructions are given under "To Set File Permissions for the Server" on page 60.

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

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

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

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

3.3 How to Use Network Resource Strings

Once you have chosen the client 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:

(C shell)

```
% waitstone \!@oboe\!gemStoneName -1
```

or (Bourne or Korn shell)

```
$ 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.

The list of command line GemStone commands in Appendix B, "GemStone Utility Commands," tells which options of each command can be specified as an NRS. Besides location, an NRS can describe the network resource type so GemStone can more accurately interpret the command line. Sometimes an NRS can also include your authorization to use that resource. See Appendix C, "Network Resource String Syntax," for more information.

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:

(C shell)
```
% setenv GEMSTONE_NRS_ALL #dir:/user2/apps/logs
```

or (Bourne or Korn shell)
```
$ GEMSTONE_NRS_ALL=#dir:/user2/apps/logs
$ export GEMSTONE_NRS_ALL
```

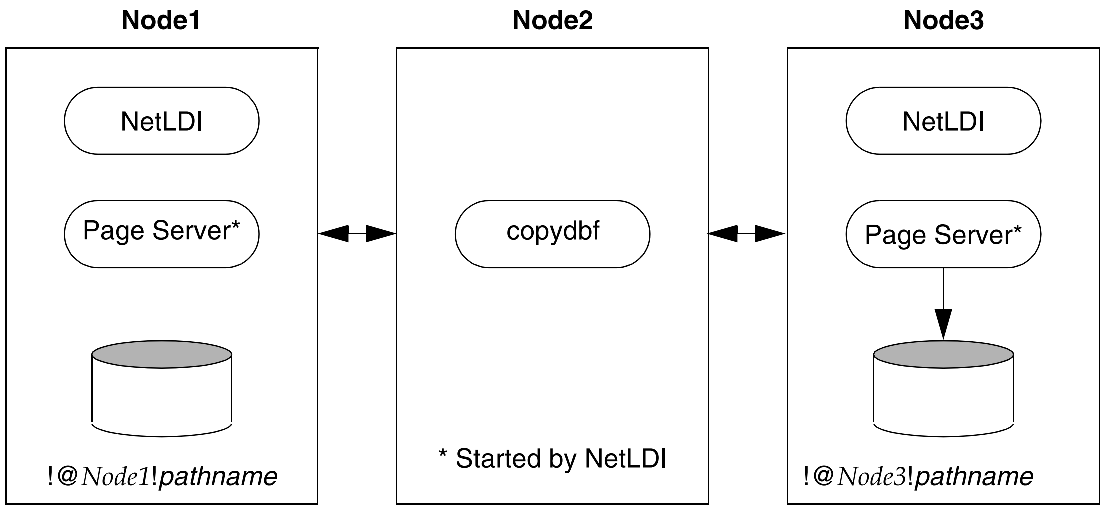
For further information about the modifiers and templates available, see Appendix C, "Network Resource String Syntax."

To Use copydbf Between Nodes

Figure 3.3 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 3.3 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 (C shell):

```
Node2% copydbf \!@Node1\!filePath \!@Node3\!filePath
```

If you use the Bourne shell, the backslashes (\) are not necessary.

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 client node must have its kernel configured for shared memory similarly to how it is configured on the server node.
- Only server nodes need a GemStone key file, not client nodes.
- 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` 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 server node and 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 tape or mount the directory on the server node that contains $GEMSTONE. Each approach is described below. 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 client node, we recommend that you also run `$GEMSTONE/install/installgs.` In particular, you should make the same selections regarding the ownership and group for the GemStone files as you did on the server node. You can save disk space later by deleting the two copies of the initial repository (`$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 client 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/GemStone3.0`.

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

(C shell)

```
Client% setenv GEMSTONE /Server/users/GemStone3.0
Client% source $GEMSTONE/bin/gemsetup.csh
```

or (Bourne or Korn shell)

```
$ GEMSTONE=/Server/users/GemStone3.0
$ export GEMSTONE
$ . $GEMSTONE/bin/gemsetup.sh
```

Step 2. If they do not 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

The following examples expand on Chapter 2 by showing the additional processes that are necessary in a distributed configuration. Because the AIO page server and GcGem on the Stone's node are not important to the networking discussion, they are omitted from the illustrations in this chapter.

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.

Examples below illustrate the following distributed applications:

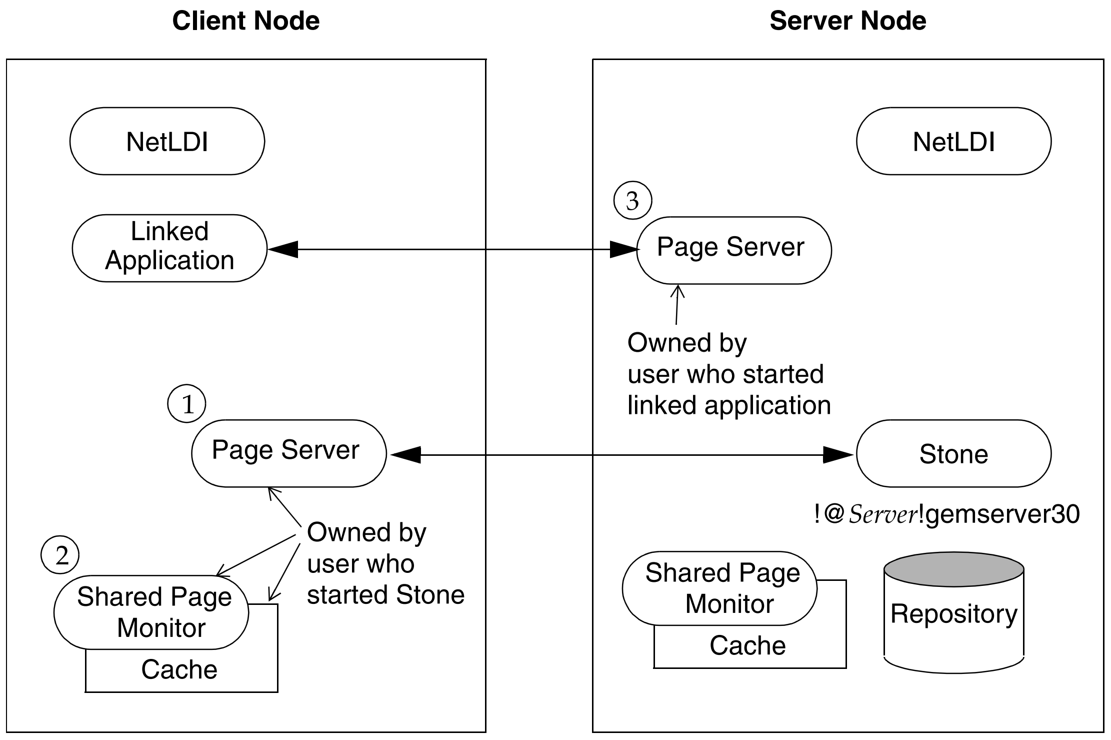
- a linked application connected to a Stone on another node,
- an RPC application with both the session process and the Stone on the server node,
- an RPC application with the session process on the application's node, and
- an RPC application in which all three are on different nodes.

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 3.4 shows how a linked application on a client node communicates with a Stone and repository on the 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 3.4 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 client 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 client 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 server. There will be one such page server process on the 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 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 84.

The following steps set up a linked application on the client node. They use software in an NFS-mounted installation on the server node; that directory is already mounted on the client node as `/Server/GemStone3.0`.

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

(C shell)

```
Client% setenv GEMSTONE /Server/GemStone3.0
Client% source $GEMSTONE/bin/gemsetup.csh
```

or (Bourne or Korn shell)

```
$ GEMSTONE=/Server/GemStone3.0
$ export GEMSTONE
$ . $GEMSTONE/bin/gemsetup.sh
```

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

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

Step 3. Start a NetLDI on the client 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):

```
Client% 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:

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

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

```
Client% topaz -l
topaz> set gemstone !@Server!gemserver30
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 in Figure 3.5), 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 a Gem network object (`gemnetobject` or `gemnetobjcsh`) 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 98.

Step 1. 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.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. There are three choices:

❒ 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
```

❒ 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` (the default) if you are using the Bourne shell, or to `gemnetobjcsh` if you are using the C shell. 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:

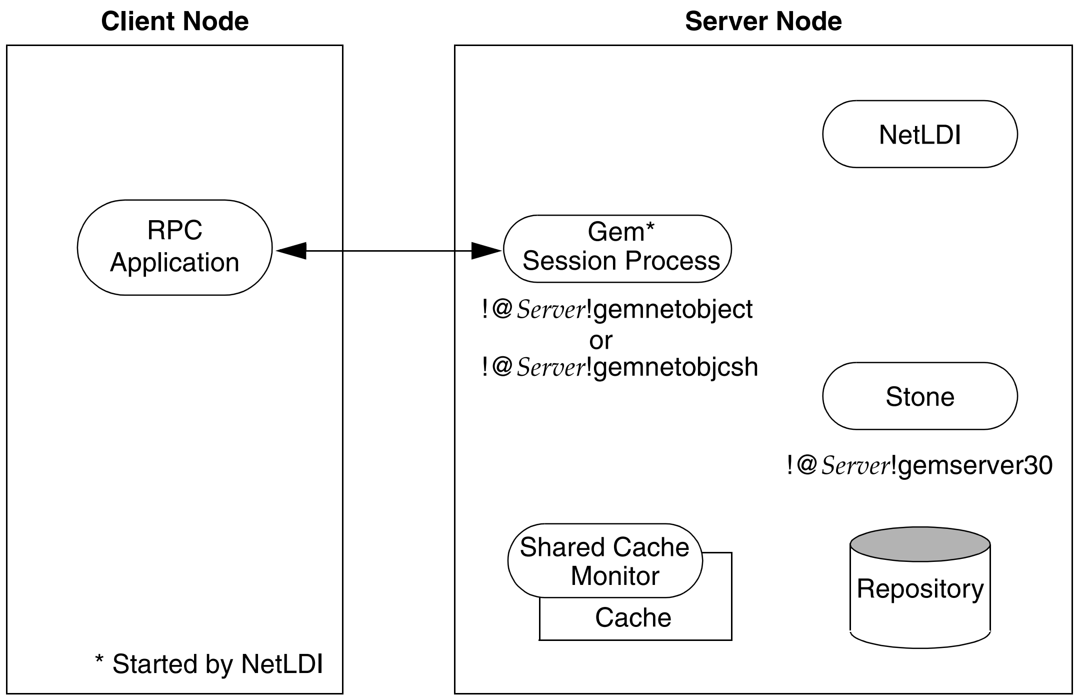
(C shell)

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

or (Bourne or Korn shell)

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

Figure 3.5 Starting a Session Process on the Server Node



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 3.6 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 112). In Figure 3.5, however, the NetLDI also starts the RPC Gem session process at the request of the application.

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 client node. There are three choices:

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

```
machine Client login userName password userPasswd
```

❒ 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 client node and start a NetLDI.

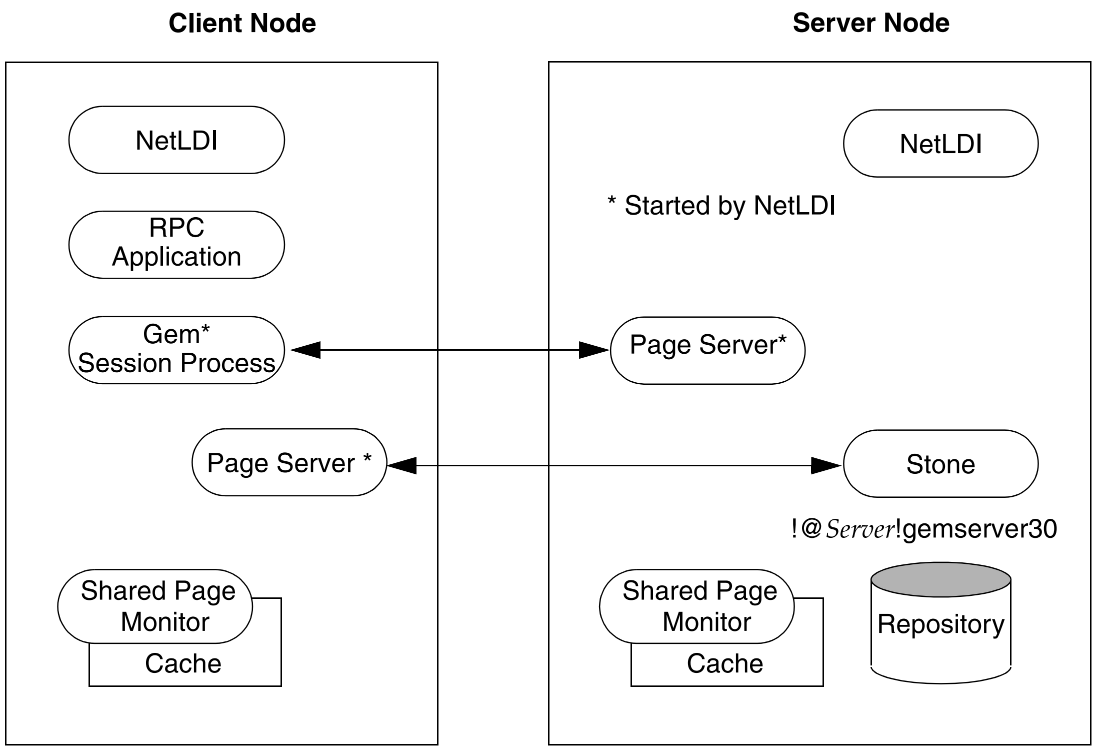
❒ To start the NetLDI for password authentication, make sure `$GEMSTONE/sys/netldid` is owned by root and has the S bit set. Issue this command (on some operating systems, you may have to issue it as root):

```
Client% 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:

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

Figure 3.6 Starting the Session Process on a Client Node



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

```
Client% topaz
```

Step 4. Set GemNetId to `gemnetobject` (the default) if you are using the Bourne shell, or to `gemnetobjcsh` if you are using the C shell. These network objects identify scripts that start a session process using your preferred shell. For example:

(C shell)
```
topaz> set gemnetid gemnetobjcsh
```

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

```
topaz> set gemstone !@Server!gemserver30
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 3.7. 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 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).

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.

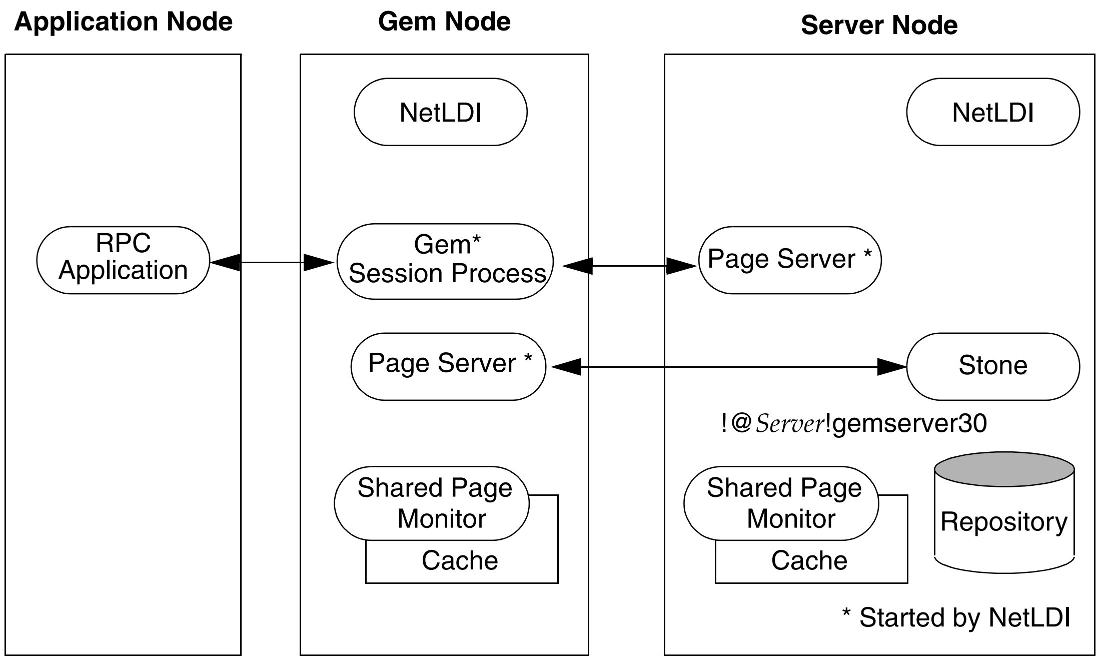
❒ You can create a `.netrc` file in the your home directory on the application node containing a line like the following, where the password is your password on the Gem's node.

```
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
```

Figure 3.7 Connecting an RPC Application, Three Nodes



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

❐ 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):

```
Client% 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:

```
Client% 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 match the shell you are using on the Gem's node, and include the location, *gemNode,* in the NRS. For example:

(C shell)

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

or (Bourne or Korn shell)

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

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

```
topaz> set gemstone !@Server!gemserver30
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 client node requires proper system configuration of the client node and frequently requires permission for network access from server to client as well as from client to server.

❐ The UNIX kernel on the client 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 3-104 and 3-106), and the directories in `/opt/gemstone` must be writable.

❐ Make sure the user who started the Stone has an account on the client node. This user also must have write permission for $HOME so that log files for the client 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 client node, try logging in on the server node as the same UNIX user account. We suggest 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 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 servio as of Tue Aug 16 14:44:45 2011
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 (the switches may be different on your system), **ps -ef | grep topaz**.

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 84.

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 a NetLDI is running on the server by invoking **gslist**; the default name is netldi30. If you need to start one, 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/netldi30.log`. Its contents may help you diagnose problems.

❐ Make sure 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 123 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 107.)

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**.

Chapter 4 Running GemStone

This chapter shows you how to perform some common GemStone system operations:

- Starting the GemStone Object Server (page 126)
- Starting Network Long Distance Information (NetLDI) servers (page 133)
- Identifying running servers (page 134)
- Logging in to a GemStone session (page 135)
- Identifying the current sessions (page 140)
- Shutting down the object server (page 142)

Troubleshooting hints are provided in each startup and login topic.

Additional topics at the end of the chapter explain how to:

- Recover from an unexpected shutdown (page 143)
- Load objects in bulk (page 147)
- Enter and use manual transaction mode, which we recommend that you use whenever possible (page 148)

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/GemStone6.6`. 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 394.
- 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 47.

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 143).

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 `GemStone6.6-sparc.Solaris`. For example:

(C shell)
```
% setenv GEMSTONE /users/GemStone6.6-sparc.Solaris
```

or (Bourne or Korn shell)
```
$ GEMSTONE=/users/GemStone6.6-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` scripts. There is one version for users of the C shell and another for the Bourne and Korn shells. These scripts also set your man page path to include the GemStone man pages.

C shell)

```
% source $GEMSTONE/bin/gemsetup.csh
```

or (Bourne or Korn shell)

```
$ . $GEMSTONE/bin/gemsetup.sh
```

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 `gemserver66.` For additional information about **startstone**, see the command description in Appendix B.

Starting up after unexpected shutdown

If the Stone repository monitor was not cleanly shutdown, on restart it will attempt to recover automatically. If the repository extents are not corrupted, and all extent and transaction log files are available, the stone may be able to recover all committed transaction up to the point of the shutdown.

For automatic recovery to succeed, it is important that all transaction logs required for recovery are online at all times. Since long-running transaction may span multiple transaction logs, you may need to keep a number of transaction logs online before archiving.

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/gemserver66.log`), then refer to the discussions that follow this summary.

- The GemStone key file is missing or invalid (see page 128).
- The shared page cache cannot be attached (see page 128).
- An extent file is missing or cannot be opened for exclusive use because another GemStone process is using it (see page 129).

- Because of the timing of a system crash, the repository monitor is trying to create an extent that already exists (see page 129).
- An extent is supposed to have a replicate, but the replicate is missing (see page 130).
- A transaction log needed for recovery is missing, or the log directory or device does not exist (see page 131).
- The repository has become corrupted (see page 131).

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

Key File Missing or Invalid

The Stone repository monitor must be able to read a 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.
- 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 43 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 60. 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 monitor cannot access a repository extent file, it logs a message like the following:

```
startstone[Error]:    Stone process (id=6473) has died.
      reason = File      = /users/GemStone6.6/data/extent0.dbf
Error = open() failure;    DBF Op = Open;    DBF Record = -1;
UNIX Codes = errno=2, ENOENT, No such file or directory
```

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 monitor log:

```
Error = exclusive open: file is open by another process
UNIX Codes = errno=13, EACCES, Authorization failure”
Error in opening repository for exclusive access.
```

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 log:

```
An error in recovery for extentId 1:
  fileName= /users/GemStone6.6/data/extent1.dbf
Extent 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’s log file, which by default is `$GEMSTONE/data/`*gemStoneName*`.log`.

- 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.

Replicates are subject to the same consistency checks as extents.

Extent Replicate Missing

If you run your system with extent replicates and one is missing, GemStone puts an error message like this in the GemStone log:

```
Replicate is not mounted, filename = 
  !TCP@servio#dbf!/users/replicates/replica1.dbf
Error: File = /users/replicates/replica1.dbf open() failure
DBF Operation Open; (no DBF record), UNIX codes: errno=2,   ENOENT,
  No such file or directory
```

If this message appears when you try to start GemStone, replace the missing replicate. For example:

```
% copydbf extent1.dbf /users/replicates/replica1.dbf
```

where `extent1.dbf` is an extent of the repository and `replica1.dbf` is the missing replicate of the extent named in the error message. Then try to start GemStone again.

Transaction Log Missing

If GemStone cannot find the transaction log file or its replicate for the period between the last checkpoint and an unexpected shutdown, it puts a message like this in the monitor 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/GemStone6.6/data/tranlog0.dbf
Error during repository recovery
```

If the log file was archived and removed from the log directory, restore the file.

CAUTION
*The **startstone -N** option (below) should be used only to recover from a disaster that corrupts or destroys transaction logs since the last checkpoint.*

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.

Repository Failure

If a log message shows problems in an extent file, you need to consider strategies for recovery. This manual describes three ways to control the effects of repository failure: replicated extents, periodic backups, and full transaction logging. Depending on how you administer your site and on the nature of the failure, you may have the following options (which are listed in order of preference):

- If you have replicated the extents in your repository, you may be able to restore the extent by using the replicate. See “How to Recover by Using an Extent Replicate” on page 200.
- If you have GemStone backups (that is, backups made using the method `fullBackupTo:`), you can restore the repository to the state of the most recent backup. If full logging was in effect (STN_TRAN_FULL_LOGGING was set to True), objects committed by subsequent transactions can then be recovered from the transaction logs. See “How to Restore a GemStone Repository” on page 281.
- You may be able to restore the repository from operating system backups, but the results may not be satisfactory. See “Why Operating System Backups May Not Be Usable” on page 275 and “How to Restore from an Operating System Backup” on page 297.
- If you have neither replicates nor a recent backup and transaction logs for valuable data, you still may 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 227.

Other Startup Failures

- Check `/opt/gemstone/locks` and remove old files (`..LCK` or `..FIFO`) having the same Stone name. (On systems that have been running a previous release, these file may be in `/usr`.) 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 227.) If the page audit is good but GemStone still doesn’t start, check your installation configuration. For more help, call 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. There are several situations in which a NetLDI is necessary (each is described in Chapter 3):

- 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.

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 `GemStone6.6-sparc.Solaris`. For example:

(C shell)

```
% setenv GEMSTONE /users/GemStone6.6-sparc.Solaris
```

or (Bourne or Korn shell)

```
$ GEMSTONE=/users/GemStone6.6-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 C shell and another for the Bourne and Korn shells. These scripts also set your man page path to include the GemStone man pages.

(C shell)

```
% source $GEMSTONE/bin/gemsetup.csh
```

or (Bourne or Korn shell)

```
$ . $GEMSTONE/bin/gemsetup.sh
```

Step 3. Start the NetLDI by using the **startnetldi** command.

❒ 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):

```
Client% 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:

```
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 102.

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 (the default is `/opt/gemstone/log/`*netLdiName*`.log`) for clues.

- 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` and in the *GemStone Programming Guide*.

4.3 To List Server Status

The **gslist** utility lists all Stone repository monitors, shared page cache monitors, and NetLDIs that are running. The **--v** option causes it to verify that each process is alive and responding.

The **gslist** command checks the locks directory (`/opt/gemstone/locks`) for entries.

```
% gslist -v
Status  Version    Owner     Started      Type   Name
------ --------- --------- ------------ ------ ----
  OK    6.6.0     gsadmin   Dec 13 11:49 cache  gemserver66@mozart.ge
mstone.com
  OK    6.6.0     gsadmin   Dec 13 11:49 Stone  gemserver66
  OK   6.6.0     gsadmin   Dec 13 11:54 Netldi netldi66
```

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. Additional information about the GemStone administrative logins is given in Chapter 5, "User Accounts and Security." Additional information about logging in from a remote node is given in Chapter 3, "Connecting Distributed Systems."

Two examples follow a brief discussion of environmental variables. 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 80.

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 `/users/GemStone6.6-sparc.Solaris.` The directory `$GEMSTONE/bin` should be in your search path for commands. The next topic contains an example.
- 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 "How GemStone Uses Configuration Files" on page 394.

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 126 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 *gemserver66*.

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 `GemStone6.6-sparc.Solaris`. For example:

(C shell)

```
% setenv GEMSTONE /users/GemStone6.6-sparc.Solaris
```

or (Bourne shell)

```
$ GEMSTONE=/users/GemStone6.6-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, and
GEMSTONE_LANG.

Step 2. Set your UNIX path. One way is to use one of the `gemsetup` scripts. There is one version for users of the C shell and another for users of the Bourne and Korn shells. These scripts also set your man page path to include the GemStone man pages.

(C shell)

```
% source $GEMSTONE/bin/gemsetup.csh
```

or (Bourne or Korn shell)

```
$ . $GEMSTONE/bin/gemsetup.sh
```

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 *gemserver66* 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 `netldi66`. (This list also shows the Stone and shared page cache monitor.)

```
% gslist
Status  Version    Owner      Started    Type   Name
------ --------- --------- ------------ ------ ----
exists 6.6.0     gsadmin   Dec 14 08:42 cache  gemserver66@node3
exists 6.6.0     gsadmin   Dec 14 08:42 Stone  gemserver66
exists 6.6.0     gsadmin   Dec 14 08:42 Netldi netldi66
```

If necessary, start a NetLDI:

❒ 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):

```
Client% 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:

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

Step 2. 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. There are three choices:

- ❒ You can create a `.netrc` file in the 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 3. Start the RPC application (such as Topaz), then set the UserName.

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

Step 4. If you are using the C shell, set GemNetId (the name of the Gem service to be started) to `gemnetobjcsh`. (The default, `gemnetobject`, is for the Bourne shell.) These scripts start the separate Gem session process for you, and they read your shell initialization file (`.cshrc` or `.profile`) if it exists. For example:

```
topaz> set gemnetid gemnetobjcsh
```

Step 5. 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 420) can restrict the number of logins to fewer than a particular key file allows. An entry in the Stone's log file shows the maximum at the time the Stone started. By default, the Stone's 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 412) 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, 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 40. 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 39.

- The UNIX kernel file descriptor limit can restrict the number of sessions, and GemStone executables attempt to raise that limit. For information, see "Estimating File Descriptor Needs" on page 38 for the Stone and page 83 for

Gems. You can examine the kernel limit on some operating systems 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 Aug 15
17:18:23 PDT 2011
T     ID     KEY          MODE         OWNER    GROUP
Shared Memory:
m    768 0x4c177155 --rw-rw----  gsadmin     pubs
```

  Typical problems occur with **gsilnk** or similar 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 121.
- Some users of the Korn shell may encounter errors if their `.profile` contains commands that are not IEEE POSIX compliant. Such users should place those ksh commands within a conditional like the following:

```
hash -r 2>/dev/null
status=$?
if [ $status -ne 0 ]; then
   # Place Korn shell-specific initialization here
fi
```

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

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: 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 terminating it directly.*

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 users (except GcUser) have logged out. One way to find out about other users 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: DataCurator
```

Then use the **stopstone** command, which performs an orderly shutdown in which all committed transactions are written to the extent files and to any replicates. There is a similar command to shut down the NetLDI network service.

```
% stopstone [gemStoneName] [-i]
% stopnetldi [netLdiName]
```

If you do not supply the name of the repository monitor, the utility will prompt you for one. The default name during startup was `gemserver66`. If necessary, use **gslist** to find the name.

You will be asked for the name and password of a user, who must have SystemControl privileges (initially, these users are SystemUser and DataCurator). User accounts and privileges are described later in this chapter.

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.

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

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

CAUTION
*If you must kill a specific Gem session process or GemStone server processes, use **kill**, NOT **kill -9** or another uncatchable signal, because the latter may not result in a clean shut down or may cause the Stone repository monitor to shut down when you intended to kill only a Gem process. After sending **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. After sending **kill -9** to a Stone, use **ipcs** to determine whether **ipcrm** should be invoked.*

4.7 How to Recover from an Unexpected Shutdown

The system 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 one of these 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 the system shuts down unexpectedly, check the message at the end of the GemStone log file to begin diagnosing the problem. Unless you specified another file on the **startstone** command line, the GemStone log is `$GEMSTONE/data/`*gemStoneName*`.log`. This directory also contains log files for the Stone child processes: the shared page cache monitor, the AIO page server, the free frame page server, and the garbage collection session. The child processes have log names formed from *gemStoneName,* the process id, and a descriptive abbreviation. For instance:

`gemserver66.log`	Stone repository monitor
`gemserver6616937pcmon.log`	Shared page cache monitor
`gemserver6616921pgsvraio.log`	AIO page server
`gemserver6616922pgsvrff.log`	Free Frame page server
`gemserver6616923pagemanager.log`	Page Manager
`gemserver6616924gcgem.log`	GcGem

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:

```
[12:54:02.818]
    SHUTDOWN command was received from:
               User: DataCurator
           Gem Host: 10.80.10.59
            Gem PID: 14431
         GCI Client:
            Session: 6

[12:54:02.969]
    Now stopping GemStone.
```

After a normal shutdown, restart GemStone in the usual manner. For instructions, see “How to Start the GemStone Server” on page 126 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/GemStone6.6/data/extent0.dbf
PageId = 94
File = /users/GemStone6.6/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 299.

Shared Page Cache Error

If you find a message similar to the following in the GemStone log, the shared page cache monitor process (`shrpcmonitor`) died.

```
The stone's connection to the local shared cache monitor 
was lost.
    Error Text: 'Network partner has disconnected.'
```

The monitor log, `$GEMSTONE/data/`*gemStoneName*`_pcmon`*nnnn*`.log`, may indicate the reason.

Check `/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 `gemserver66@127.0.0.1` and `gemserver66@127.0.0.1..LCK` for the default Stone name gemserver66.

The unexpected shut down 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 126.

Fatal Error Detected by a Gem

If a Gem session process detects a fatal error that would cause it to halt and dump a core image, the Stone repository monitor may do the same when it is notified of the event. This response on the part of the Stone is configurable through the STN_HALT_ON_FATAL_ERR option. 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_ERR 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 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 227).

   If the page audit fails, read the advice under "Disk Failure or File System Corruption" on page 144 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 126.
3. If the restart fails, you may have to restore the repository (see "How to Restore a GemStone Repository" on page 281).

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, or the Stone was stopped with a kill -9. In that event, the Stone repository monitor automatically recovers committed transactions the next time it starts. Use **startstone** to restart GemStone. For instructions, see "How to Start the GemStone Server" on page 126." See Appendix B for more information about the **startstone** command; or, for on-line documentation, type **startstone -h** or **man startstone**.

4.8 Diagnosing Problems with Running Sessions

Sessions may appear to be unresponsive while executing long-running server operations. If GemStone processes appear to be unresponsive for no known reason, you should work with GemStone Technical Support to diagnose any potential problems.

Sending the signal SIGUSR1 to a GemStone process;

```
% kill -USR1 <pid>
```

will cause both the GemStone Smalltalk code stack and the internal C stack to be printed to stdout. For the stone or an RPC gem, this is the processes' log file. For linked topaz, this is the topaz output.

4.9 How to Bulk-Load Objects

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

- Decrease the GEM_TEMPOBJ_CACHE_SIZE and increase GEM_PRIVATE_PAGE_CACHE_KB configuration options. Then divide the loading operation into several transactions. Try to keep the size of each transaction (the number of 8 KB pages written) somewhat smaller than the combined size of the Gem's private page cache and the shared page cache—perhaps 1/2 to 2/3 of their combined size. Limiting the transaction size reduces the time required for each commit operation.
- 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`.
- Disable epoch garbage collection and garbage collection of your Gem's not-connected object set. These steps save the CPU time ordinarily devoted to scanning for dereferenced objects. To do this, log in as GcUser and set `#epochGcEnabled` to False. From the session performing the bulk load, set the runtime parameter `#NotConnectedThreshold` to a large value, such as the maximum SmallInteger; the procedure is described on page 86.
- 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 426 and 426, respectively.

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

4.10 Considerations for Large Repositories

This section presents special considerations that apply to large repositories and repositories with a large number of sessions.

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 442.

Using remote caches

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.

- Mid-level caches can reduce network traffic for remote sessions. See "Using Mid-Level Caches" on page 45.
- To improve performance on remote caches, set GEM_PGSVR_UPDATE_CACHE_ON_READ (page 409) to True so that remote Gem sessions will update their local caches. For example:

  ```
  GEM_PGSVR_UPDATE_CACHE_ON_READ = TRUE;
  ```

4.11 Transaction Mode and Disk Space

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_SIGNAL_ABORT_CR_BACKLOG, the Stone repository monitor signals the session to abort. 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 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
%
```

Even in manual transaction mode, it is possible to cause a commit record backlog, depending on how your system is configured. Sessions that are idle should either issue regular aborts, or set up signalAbort handlers to abort when requested by the Stone.

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.

Chapter
5

User Accounts and Security

This chapter describes GemStone user accounts, and shows you how to perform some common GemStone user administration tasks:

- Describes User accounts in GemStone.
- 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*.
- How to configure GemStone login security by restricting valid passwords, imposing password and account age limits, and monitoring intrusion attempts.

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 of privileges and Segments, see the chapter of the *GemStone/S Programming Guide* that discusses security.

5.1 GemStone Users

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. This UserProfile object contains information describing objects that the user is allowed to examine or modify, privileges that the user has to perform certain operations, and security information.

Each UserProfile has the following information:

User ID	A unique String that identifies the user to the GemStone system.
Password	The GemStone-specific password (an InvariantString) to use to validate logins. GemStone stores the password in encrypted form in a secure manner.
Default Segment	An instance of Segment that determines the default read and write authorizations for objects created by the user.
Privileges	A collection of symbols that allow the user to perform certain "privileged" system functions.
Groups	In conjunction with segments, group membership is used to allow access to restricted objects for specific categories of users.
SymbolList	The list of SymbolDictionaries that this user can see.

These are discussed in more detail starting on page 154.

Other information related to the user account is stored in an instance of UserSecurityData; this includes data related to security features. Instances of UserSecurityData are private and protected, but some information, such as lastLoginTime, can be accessed via methods in UserProfile.

AllUsers

Each instance of UserProfile must be in the global collection, AllUsers. AllUsers is the single instance of UserProfileSet. AllUsers acts as the "root" for all objects in the repository; any object in the repository must be reachable from AllUsers, usually via the SymbolLists of the UserProfiles, otherwise it is subject to garbage collection.

Special System Users

When GemStone is first installed, AllUsers has UserProfiles already defined for the following users. These are the special system users. *You must never delete these users.* These users may not have privileges removed, cannot be disabled, and their accounts are not subject to password or account age limits.

SystemUser

SystemUser is analogous to `root` in UNIX. SystemUser has all privileges, belongs to all predefined groups, and is authorized to read and write all objects regardless of segment protection. These privileges cannot be taken away, so SystemUser can always write to all objects. This account is used only to perform GemStone system upgrades, modify some system configuration settings, and other special-purpose operations that must be highly restricted.

The SystemUser account is the owner of the SystemSegment, which contains the kernel classes.

WARNING
Logging in to GemStone as SystemUser is like logging in to your workstation as root: an accidental modification to a kernel object can cause a great deal of harm. Use the DataCurator account for system administration functions except those that ***require*** *SystemUser privileges, such as a repository upgrade.*

DataCurator

The DataCurator account is the account that is normally used for day-to-day administration tasks. Initially, DataCurator is granted all privileges and belongs to all predefined groups. All GemStone UserProfiles are protected by the DataCuratorSegment.

GcUser

The GcUser account is a special account that logs in to the repository automatically to perform garbage collection tasks. You normally would only login as GcUser in order to update configuration parameters stored in GcUser’s UserGlobals.

Nameless

The Nameless account is a special account for use only by other GemStone products. Do not use this account or change it unless instructed to do so by GemStone Technical Support.

UserProfile Data

Each user profile has the following instance variable data, either explicitly specified during instance creation, or provided with default values. This information can be updated.

The requirements for updating this information vary. In many cases, the requirements are different between updating information for another user and for updating your own information. See the update methods for details.

User ID

Each UserProfile is created with a unique user Id String. Embedded spaces are permitted.

UserId may only be changed by SystemUser, and the userIds of special system users cannot be changed.

Password

Each UserProfile is created with an initial password, an Invariant String, which must not be the same as the User Id and may not be longer than 1024 characters. The user supplies this password for identification purposes at login.

Users must have the explicit privilege `#UserPassword` to change their own passwords, and there are a number of ways to constrain the choice of passwords. See the section starting on page 173 for details.

To change the password of a user other than yourself, the `#OtherPassword` privilege is required, a different method is used, and password constraints do not apply.

Default Segment

A users' default segment determines the default read and write authorizations for objects created by the user. When you add a new user to the GemStone system, you can use protocol that creates a new segment, or specify an existing segment for the user.

For more information on how segments are used to control read and write authorization for objects in the repository, see the chapter in the *GemStone/S Programming Guide* that discusses security.

To modify the defaultSegment of any user, you must have write access to the segment of that UserProfile. In addition, to modify your own defaultSegment, you must have the `#DefaultSegment` privilege.

Privileges

When you create a new UserProfile, you determine whether the new user may perform certain "privileged" system functions. For example, stopping another session or the repository itself requires a particular privilege to do so. Table 5.1 describes the types of functions that each privilege controls.

Note that privileges are more powerful than segment authorization. Although the owner of a segment can always use authorization protocol to restrict read or write access to objects in a policy, an administrator with appropriate privileges, such as DataCurator, can override that protection by sending privileged messages that let you change the authorization scheme.

Table 5.1 GemStone Privileges

Type of Privilege	Privileged Operations
`SystemControl`	`#SystemControl` is required by methods that start or stop sessions, including operations that invoke the multi-threaded scan (see page 339); for methods that suspend or resume logins, send signals to other sessions, and manage checkpoints.
`Statistics`	`System class >> stoneStatistics`
`SessionAccess`	`#SessionAccess` privilege is required to find out information about sessions other than the current session, or to perform operations on other sessions.
`UserPassword`	Required to change your own password using `UserProfile>>oldPassword:newPassword:`
`DefaultSegment`	This privilege is required to set a UserProfile's default Segment using `UserProfile>>changeToSegment:` or a method that invokes that.

Table 5.1 GemStone Privileges (Continued)

Type of Privilege	Privileged Operations
`OtherPassword`	You must have `#OtherPassword` privilege to make any changes to a UserProfile. This includes adding or removing a SymbolDictionary to/from a SymbolList that is not your own. `#OtherPassword` is also required to find out information about UserProfiles other than the currently logged in session. `#OtherPassword` is required to make any changes to AllUsers, including creating a new user and configuring security requirements.
`SegmentCreation`	Required in order to creating a new segment, using `Segment class>>newInRepository:` or any methods that invoke this.
`SegmentProtection`	You must have `#SegmentProtection` to update the authorizations of a segment, other than one that is owned by the current session's user. This includes `Segment>>group:authorization:`, `ownerAuthorization:`, and `worldAuthorization:`.
`FileControl`	`#FileControl` is required for operations that access external files, including operations related to backup, restore, transaction logs, and extents.
`GarbageCollection`	Required to perform any garbage collection operation, to start and stop Admin and Reclaim GcGems, and force epoch or reclaim to run. Also required to audit and profile the repository.

Groups

GemStone uses group membership to facilitate access to objects that are protected by segments. While certain objects must be protected from read or write access by other users in the system (the "world"), you may still need to grant access to specific individual users. By adding group authorization to the segment that protects these objects, and adding the corresponding group membership to that user, you can provide a user with the appropriate access to these objects.

A segment can authorize multiple groups and a user can be a member of multiple groups. There are several predefined groups, as shown in Table 5.2. By default, all new users become members of group Subscribers.

AllGroups

AllGroups is a global collection of Strings, that includes all groups defined for all users and all security policies.

Initially, AllGroups contains the following:

Table 5.2 GemStone Groups

Group name	Access
System	Members of this group have write access to objects protected by the GcUser's Segment.
Publishers	Members of this group have write access to objects protected by **PublishedSegment**.
Subscribers	Members of this group have read access to objects protected by **PublishedSegment**.

Symbol Lists

As explained in the *GemStone/S Programming Guide*, the GemStone Smalltalk compiler follows a well-defined path in resolving objects named by source code symbols. First, the compiler considers the possibility that a variable name might be either local (a temporary variable or an argument) or defined by the class of the current method definition (an instance variable, a class variable, or a pool variable). If a variable is none of these, the compiler refers to an Array of SymbolDictionaries in the user's UserProfile and current session state. That Array is called the user's *symbol list*. The symbol list tells Smalltalk which of many possible GemStone SymbolDictionaries to search for an object named in a Smalltalk program.

For each 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, refer to the *GemStone/S Programming Guide*.

New UserProfiles are created with the following SymbolDictionaries, in this order:

UserGlobals Each UserProfile has its own SymbolDictionary for the user's private symbols. UserGlobals includes the key `#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 *GemStone/S Programming Guide* manual.

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

Published The third and final element in each user's initial symbol list is a SymbolDictionary for application objects that are "published" to all users. Users who are members of the group Publishers can place objects in this dictionary to make them visible to other users. Using the Published dictionary lets you share these objects without having to put them in Globals, which contains the GemStone kernel classes, and without the necessity of adding a special dictionary to each user's symbolList instance variable.

Although all users automatically share access to objects in Globals, sharing application objects between users requires that the objects be in a SymbolList that is visible to both users. There are three primary ways to do this:

- As a member of group Publishers, you can add the objects to the Published dictionary. This dictionary is already in each user's symbol list, so whatever you add becomes visible to users the next time they obtain a fresh transaction view of the repository. You may do this by sending the message `Published at:` *aKey* `put:` *aValue*.
- You can define a special SymbolDictionary, and add that to the user's SymbolList. The procedure is described under ""Add a Dictionary to a Symbol List" on page 165".

- 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 *GemStone/S Programming Guide*.

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

5.2 Creating and Administering User Accounts

Methods that create UserProfiles add the new UserProfile to AllUsers, the global collection (UserProfileSet) of users. UserProfiles that are not in AllUsers cannot log in.

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. This is described on "How to Start a GemStone Session" on page 135.

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 ] .
%
```

Add a User

Privileges required: write authorization in the DataCurator Segment; SegmentCreation.

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. This is described in the GemStone Installation Guide.

GemStone Smalltalk provides several 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 default segment, and the privileges and groups for that segment.

Without Privileges or Groups

You can use the simplified form to create a new user with no privileges or group memberships. A new segment is created for the user and is assigned as that user's default segment. (The new user may subsequently want to restrict access to the new segment, which is created with read permission for the world.)

In the following example, you must supply the new user's userId and password (each as a String). The password may not be the same as the UserId and may not be longer than 1024 characters.

```
topaz 1> printit
AllUsers addNewUserWithId: 'theUserId'
      password: 'thePassword'.
"commit the new UserProfile"
System commitTransaction
%
```

Assign Privileges and Group Memberships

Using the complete form allows you to assign privileges to the new user, add the user to groups (used for read/write authorization in segments), and explicitly specify the user's default segment.

Execute this expression (an example follows):

```
topaz 1> printit
AllUsers addNewUserWithId: 'theUserId'
      password: 'thePassword'
      defaultSegment:
            (Segment newInRepository:
                  (System segment repository))
      privileges: anArrayOfPrivStrings
      inGroups: aCollectionOfGroupStrings  .

(AllUsers userWithId: 'theUserId') defaultSegment
      owner: (AllUsers userWithId: 'theUserId') .
"commit the new UserProfile"
System commitTransaction.
%
```

You must supply the new user's UserId and Password, and specify any privileges or group memberships. The password may not be the same as the UserId and may not be longer than 1024 characters. A new segment is created for the new user, and is assigned as that user's default segment. The `owner:` message establishes the new user as the owner of that default segment, thus allowing the new user to log in to GemStone. For example:

```
topaz 1> printit
AllUsers addNewUserWithId: 'Mary'
      password: 'herPasswd'
      defaultSegment:
            (Segment newInRepository:
                  (System segment repository))
      privileges: #( 'UserPassword' 'DefaultSegment' )
      inGroups: #[ 'MarathonRunners' ] .

 (AllUsers userWithId: 'Mary') defaultSegment
      owner: (AllUsers userWithId: 'Mary') .
"commit the new UserProfile"
System commitTransaction.
%
```

Alternatively, you can assign an existing segment to a user by explicitly specifying a `defaultSegment:` argument. Note that before he or she can log in to

GemStone, the new user must be authorized to read and write in the specified default segment. To modify the previous example:

```
        ...
        defaultSegment: anExistingSegment
        ...
```

You can subsequently refer to the new user's default segment symbolically by executing an expression of the form:

```
topaz 1> printit
|theUser|
theUser := AllUsers userWithId: 'theUserId' .
UserGlobals at: #aSymbol put: (theUser defaultSegment)
%
```

For more information about privileges and default segments, see the sections "Privileges" and "Default Segment" on page 154.

Change Your Own Password

Privileges required: UserPassword.

Your choice of passwords for your own account may be subject to optional restrictions as to pattern and the use of certain words. For information, ask your data curator or see "GemStone Login Security" on page 173.

To modify your own GemStone password, execute the following expression. The new password will take effect when you commit the current transaction. The password may not be the same as the UserId and may not be longer than 1024 characters.

```
topaz 1> printit
System myUserProfile
        oldPassword:'oldPasswordString'
        newPassword:'newPasswordString' .
System commitTransaction
%
```

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 optional pattern restrictions described on page 173 because it can only be set by a user having the OtherPassword privilege. The password may not be the same as the UserId and may 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.

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 authorization 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. Refer back to Table 5.1 on page 155 for the Smalltalk functions that are controlled by each privilege.

Assign a Privilege to a User

Privileges required: write authorization in the DataCurator Segment.

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
%
```

Here's an example that assigns three new privileges to user Bob:

```
(AllUsers userWithId: 'Bob')
      addPrivilege: 'SystemControl';
      addPrivilege: 'SessionAccess';
      addPrivilege: 'UserPassword' .
```

Revoke a User's Privilege

Privileges required: write authorization in the DataCurator Segment.

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
%
```

The following example revokes three of user Jane's privileges:

```
(AllUsers userWithId: 'Jane')
      deletePrivilege: 'SystemControl';
      deletePrivilege: 'SessionAccess';
      deletePrivilege: 'UserPassword' .
```

Redefine a User's Privileges

Privileges required: write authorization in the DataCurator Segment.

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

To redefine privileges 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' 'DefaultSegment') .
System commitTransaction
%
```

Add a Dictionary to a Symbol List

Privileges required: write authorization in the Segment of the symbol list being modified, which typically is that user's default Segment.

You can add a dictionary to a user's persistent 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) in to the user's own symbol list:

```
topaz 1> printit
System myUserProfile
      insertDictory: 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.

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.

Add a User to a Group

Privileges required: write authorization in the DataCurator Segment.

Do the following to add a user to a group:

```
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'
.
%
```

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.) Now, the user can read or modify any objects stored in segments for which the group MarathonRunners has the appropriate authorization.

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.

Remove a User from a Group

Privileges required: write authorization in the DataCurator Segment.

You can execute an expression of the following form to remove a user from a group:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId') removeGroup: #Sprinters.
%
```

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, and about GemStone's authorization mechanism in general, see the "Security" chapter of *GemStone/S Programming Guide*.

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.

Remove a User Group

Privileges required: write authorization in the DataCurator Segment.

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 aSegment |
theGroup := aGroupString .
"Does any segment still have authorization for this group?
If so, return the segment and exit."
SystemRepository do:
      [ :aSegment |
       (aSegment authorizationForGroup: theGroup) == #none
       ifFalse: [ ^ aSegment asString ].
      ].
"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 okay to remove the group itself”
AllGroups remove: theGroup .
%
```

Modify Someone’s User ID

Privileges required: be SystemUser, and write authorization in the DataCurator Segment.

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.

Remove an Account

Privileges required: write authorization in the DataCurator Segment.

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 segments or other objects owned by the former user, you can refer to (OldUsers userWithId: '*theUserId*') wherever you would refer to an active UserProfile.

5.3 Administering Segments

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 6 in Repository SystemRepository
Owner SystemUser write
Group Subscribers read
Group Publishers write
World none
```

Change the Authorization of a Segment

Privileges required: SegmentProtection or be the segment's owner.

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 when you commit 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.

Remove a Group from a Segment's Authorization List

Privileges required: SegmentProtection or be the segment's owner; 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
%
```

Change a User's Default Segment

Privileges required: DefaultSegment to change your own; write authorization in the DataCurator Segment to change another's.

Changes to a segment's authorization do not take effect until the current transaction is committed.

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.

Check a Segment for Authorization Errors

If your application is experiencing unexplainable authorization errors, do an object audit and examine the audit report for clues. (For information, see “How to Audit the Repository” on page 227.”)

If the audit report does not indicate inconsistencies in the repository, you can perform a segment-level consistency check, which verifies that every segment is owned, that the owner of each segment is a member of AllUsers, and that each group is an element of AllGroups. To perform the consistency check, execute the following expression:

```
topaz 1> printit
| result |
result := Array new.
SystemRepository do:[ :aSegment |
  aSegment owner == nil
  ifTrue:[
    result add: #[ aSegment asString, 'has no owner' ] ].
  ( AllUsers includes: aSegment owner )
  ifFalse:[
    result add: #[ aSegment, 'owner not in AllUsers' ] ].
  aSegment groups do:
    [ :aGroup | ( AllGroups includes: aGroup )
      ifFalse:[
        result add:
          #[ aSegment, aGroup,'group not inAllGroups']
        ]
      ]
    ].
^result
%
```

If the size of the result is **not** zero, contact your GemStone customer support representative.

5.4 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 may not be the same as the UserId and may 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 logs certain administrative changes to these security features in the Stone's system log. You may want to restrict access to that file.

The SystemUser, DataCurator, and GcUser accounts are never disabled by the security features.

Constrain the Choice of Passwords

Privileges required: write authorization in the DataCurator segment.

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 5.3 shows the messages by which you can set the pattern constraints. Send each message 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 5.3 Ways to Constrain the Password Pattern

Message to AllUsers	Comments
`minPasswordSize:` *aPositiveInteger*	Sets the minimum number of characters in a new password; *0* means no constraint.
`maxPasswordSize:` *aPositiveInteger*	Sets the maximum number of characters in a new password; *0* disables the constraint. (The password String itself may not be longer than 1024 characters.)
`maxRepeatingChars:` *aPositiveInteger*	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:` *aPositiveInteger*	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:` *aPositiveInteger*	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 a sending its corresponding Accessing message (that is, without the colon or argument shown in Table 5.3). For example, to determine the current minimum size for a password:

```
topaz 1> printit
AllUsers minPasswordSize
%
6
```

Disallowing Particular Passwords

Privileges required: write authorization in the DataCurator segment.

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 the 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: write authorization in the DataCurator segment.

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
%
```

Require Periodic Password Changes

Privileges required: write authorization in the DataCurator segment.

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, 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 as explained on page 162.

Providing Warning of Password Expiration

Privileges required: write authorization in the DataCurator segment.

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
%
08/10/2011 11:28 am
```

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
%
08/10/2011 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. Recording the lastLoginTime requires a commit on login.

Disable Inactive Accounts

Privileges required: write authorization in the DataCurator segment.

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, and GcUser accounts are not disabled by this mechanism.

DataCurator or another user with the OtherPassword privilege can reactivate the account by giving it a new password as explained on page 162.

If your repository has been in use for some time when this option is first enabled, and lastLoginTime was not previously recorded for user logins for another security features (see “Finding Out When an Account Last Logged In” on page 178), then turning on this option may disable existing accounts. You can avoid this by explicity settting the lastLoginTime to nil. For example,

```
AllUsers do: [:aUser | aUser lastLoginTime: DateTime now].
```

Make an account immune from Inactive Account Disable

Individual user accounts can be configured to avoid inactive account checks that are enabled for the repository as a whole. This feature both avoids a commit on login to updat e the lastLoginTime, and does not disable accounts that do not log in for a period longer than the repository’ssetting for **staleAccountAgeLimit**.

```
topaz 1> printit
(AllUsers userWithId: 'audit') disableStaleAccountChecks .
System commitTransaction
%
```

Stale account checks can be re-enabled (according to the other settings in the repository) using **enableStaleAccountChecks**:

```
topaz 1> printit
(AllUsers userWithId: 'audit') enableStaleAccountChecks .

System commitTransaction
%
```

Disable a User’s Account

Privileges required: write authorization in the DataCurator segment.

You can have the system disable an individual user’s account. To do this, send the message `disable` to the account’s UserProfile. This example disables the account “qa2”:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') disable .
System commitTransaction
%
```

DataCurator or another user with the OtherPassword privilege can reactivate the account by giving it a new password, using the `password:` method. For details, see page 162.

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 page 162 and page 162). 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, as explained on page 162.

The SystemUser, DataCurator, and GcUser accounts are not disabled by this mechanism.

Limit Concurrent Sessions by a Particular UserId

Privileges required: OtherPassword.

You can limit the number of concurrent sessions logged in under a particular userId 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.

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 10 minutes. The default limits can be changed by setting the STN_LOG_LOGIN_FAILURE_LIMIT and STN_LOG_LOGIN_FAILURE_TIME_LIMIT configuration options.

The log message gives the following information:

```
---Fri 05 Aug 2011 09:39:40 PDT   ---
   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). Be default, the account is disabled when the number of failures exceeds 15 within 15 minutes. The default limits can be changed by setting the STN_DISABLE_LOGIN_FAILURE_LIMIT and STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT configuration options.

Subsequent attempts to login as that account result in the following error message:

```
Login failed: the GemStone userId/password combination is
invalid.
```

The SystemUser, DataCurator, 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 on page 162.

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 described in this section:

- the password expired (through aging or a login limit),
- the account remained inactive, or
- 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 page 162.

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
```

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', 'LoginsWithInvalidPassword', or 'DisabledByAdmin'.

Chapter 6

Managing Repository Space

The *repository* is the logical unit that represents the universe of shared objects that are stored within a GemStone 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`.

Optionally, GemStone can maintain a *replicate* of each extent.

Whenever GemStone performs a *checkpoint,* it makes sure that transactions committed before the checkpoint have been written to the repository extents and any extent replicates. The STN_CHECKPOINT_INTERVAL configuration option 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 70.

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 space in the repository that is currently free,
- how to create more space by adding an extent (and optional extent replicate) while the repository is in use,
- how to remove an extent or an extent replicate,
- how to reallocate objects among extents,
- how to replace a corrupted extent with the extent replicate, and
- how to recover from an error caused by a full disk.

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, always in increments of 1 MB:

- 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 indexes, 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.)
- Objects larger than 8 KB (or 2000 OOPs) are stored directly in pages even though they may become unreferenced by the time the transaction is committed. Other temporary objects sometimes are swept onto the disk to provide additional working space in a session's memory allocation, or are explicitly moved to the disk by a Smalltalk message.
- Each session requires 0.5 MB of *headroom*. 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 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.

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. For example:

```
topaz 1> run
SystemRepository fileSize
%
5242880
```

The result of the message `fileSize` is the total size of the repository in bytes. For a single extent, it is ordinarily the same result as you would obtain by using the operating system command **ls -l** *extentName*.

```
topaz 1> run
SystemRepository freeSpace
%
688128
```

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 number of free pages in the extent multiplied by the page size (8 KB). This space does not include fragments on partially filled data pages.

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> run
SystemRepository fileSizeReport
%
```

produces:

```
Extent #1
   Filename = /users/extents/primaryExt.dbf
   Replicate = NONE
   File size =    10.00 Megabytes
   Space available = 1.56 Megabytes
Extent #2
   Filename = /users/extents/secondExt.dbf
   Replicate = /users/replicates/secondExt.dbf
   File size =       1.00 Megabytes
   Space available = 0.98 Megabytes
Totals
   Repository size = 11.00 Megabytes
   Free Space = 2.54 Megabytes
```

The number of free pages in the repository can also be determined from the cache statistic FreePages (see page 248). To obtain the free space, multiply FreePages by 8192.

6.3 How to Enter Single-User Mode

Privileges required: SystemControl and SessionAccess.

Certain procedures in this chapter must be carried out in *single-user mode,* that is, by a user who is the only one logged in to the repository. These procedures

- create or dispose of an extent replicate,
- repair object consistency errors in the repository,
- force reclaiming of dead objects in the repository, or
- restore the repository from a backup.

The GcUser (garbage collection) session also must be logged out during these procedures, and also during an object audit of the repository. The applicable method takes care of stopping that session. Object audits can be performed in either single- or multi-user mode, but more comprehensive checks are performed in single-user mode. See the discussion starting on page 227.

GemStone provides several methods to assist in bringing the repository monitor to single-user mode, and you can combine them to fit the needs of your system. The following steps are a suggestion:

Step 1. Suspend further logins:

```
topaz 1> run
System suspendLogins
%
```

Step 2. Give existing sessions time to finish.

Step 3. Stop any remaining sessions:

```
topaz 1> run
System stopOtherSessions
%
```

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
It may take as long as a minute for a session to terminate after you send either `stopOtherSessions` *or* `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 terminating it directly.*

Step 4. Carry out the intended procedure.

Step 5. Allow logins to resume:

```
topaz 1> run
System resumeLogins
%
```

If you do not send `resumeLogins`, the Stone repository monitor will re-enable logins automatically when you log out.

6.4 How to Add Extents and Extent Replicates

GemStone provides two ways to add extents or extent replicates:

- 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. 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 also add extent replicates this way by adding their names to DBF_REPLICATE_NAMES.
- You can add extents while the Stone is running by invoking the Smalltalk methods described next. These methods are especially useful in avoiding or resolving low disk space conditions because the change takes effect immediately. You can also add an extent replicate this way, if you are the only user logged in.

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, through Smalltalk. The following 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 47.

Possible Effects on Other Sessions

When a new extent (or extent replicate) is dynamically added to the logical repository through Smalltalk, sessions currently logged in must have access to the new extent. The possibility exists that an on-line 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 or extent replicates, you should use operating system commands to modify them as soon as possible. Such changes can be made without stopping the Stone.

The view of which files make up the logical repository is updated:

- when users commit or abort their sessions, and
- when 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 c*reates 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> run
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 Tue 23 Aug
2011 08:41:27 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. 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.

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, 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.

Repository>>createReplicateOf: named:

Privileges required: FileControl.

NOTE
You must perform this operation in single-user mode — that is, you must be the only user logged in to GemStone. See "How to Enter Single-User Mode" on page 188.

When you add an extent file using a Smalltalk method, you should also consider adding a corresponding extent replicate using the method `createReplicateOf:`*extentFilename* `named:`*replicateFilename*. For example:

```
topaz 1> run
SystemRepository
createReplicateOf: '$GEMSTONE/data/extent2.dbf'
named: '$GEMSTONE/replicates/replicate2.dbf'
%
```

If the specified extent replicate already exists, or if that extent already has a replicate under another file name, this method returns an error and the extent replicate is not created.

To avoid ambiguity and lessen the likelihood of unwelcome surprises, we recommend that you supply the full pathname as part of the file name argument. Be sure that the case in *extentFilename* matches the case in the file name itself.

The file name argument is passed directly to the underlying operating system for handling. Therefore, all environment variables known to the operating system at large or to the Stone process itself are acceptable. However, environment variables defined *only* for your application's process will not be recognized. For this reason, you may find it preferable to avoid using environment variables in the file name argument.

6.5 How to Remove Extents and Extent Replicates

This section explains how to remove extents and their replicates:

- The only way to remove an extent file is by first performing a backup and restore to move the contents of that extent to other extents. See "How to Remove an Extent."
- An extent replicate file may be removed after first removing its name from DBF_REPLICATE_NAMES and restarting the Stone or after removing it through Smalltalk (see "How to Remove an Extent Replicate") while you are the only user logged in.

How to Remove an Extent

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 GemStone full backup procedure described on page 276.

Step 2. Shut down the Stone repository monitor.

Step 3. Modify the DBF_EXTENT_NAMES configuration parameter to show the new extent structure. If the extent is being replicated, also remove the name of the extent replicate from DBF_REPLICATE_NAMES.

Step 4. Restore the repository from your full backup using the GemStone restore procedure described on page 281.

How to Remove an Extent Replicate

Privileges required: FileControl, SessionControl, and SessionAccess.

NOTE
You must perform this operation in single-user mode – that is, you must be the only user logged in to GemStone. See "How to Enter Single-User Mode" on page 188.

If an extent has a replicate, you can discontinue replication at run time by this procedure:

Step 1. Bring the repository monitor to single-user mode.

Step 2. Send the message `disposeReplicate:` to the repository:

```
topaz 1> run
SystemRepository disposeReplicate: 'replicateFilename'
%
```

Step 3. Exit from single-user mode.

At this point it is safe to remove the file containing the extent replicate.

6.6 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 GemStone full backup, then restore the backup after setting appropriate weights in the DBF_ALLOCATION_MODE configuration option.

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 GemStone full backup procedure described on page 276.

Step 2. Shut down the Stone repository monitor.

Step 3. Modify the DBF_EXTENT_NAMES configuration parameter to show the new extent structure. (If you want to replicate the new extent, also add the name of its extent replicate to DBF_REPLICATE_NAMES.)

```
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 50). For example:

```
DBF_ALLOCATION_MODE = 10, 10;
```

Step 5. Restore the repository from your full backup using the GemStone restore procedure described on page 281. Those instructions tell you to replace the *existing* extent with a copy of a fresh one. Do not copy anything to the location of the *new* extent; the Stone repository monitor will create the new extent at startup.

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. Such migration can be prevented by placing size limits on the existing extent, or by explicitly reclustering 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 Programming Guide*.

To Reallocate Objects Among the Same Number of Extents

Changes to DBF_ALLOCATION_MODE directly affect only the subsequent allocation of pages for new or modified objects. If you want to change the allocation of existing objects, perform a GemStone full backup, then restore the backup after placing appropriate size limits in the DBF_EXTENT_SIZES configuration option.

For example, suppose your existing repository contains 800 MB and you want to divide them about equally between two existing extents. To populate each extent with about 400 MB, follow this procedure:

Step 1. Back up your repository using the GemStone full backup procedure described on page 276.

Step 2. Shut down the Stone repository monitor.

Step 3. Edit the DBF_EXTENT_SIZES configuration option to limit the size of the first extent temporarily to the size you want to become. For example, if you want half of an existing 800 MB repository to remain there, set the size of that extent to 400 MB. Leave the other extent unlimited. For example,

```
DBF_EXTENT_SIZES = 400,  ;
```

Step 4. Edit the DBF_ALLOCATION_MODE configuration option to reflect the intended distribution of pages (see “Allocating Data to Multiple Extents” on page 50). This setting will determine the distribution of new or modified objects. For example:

```
DBF_ALLOCATION_MODE = 10, 10;
```

Step 5. Restore the repository from your fullbackup using the GemStone restore procedure described on page 281. Those instructions tell you to delete your existing extents, and then to replace the *first* extent listed in DBF_EXTENT_NAMES with a copy of a fresh one. Do not copy anything to the location of the *second* extent; the Stone repository monitor will create that extent at startup.

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. Such migration can be prevented by maintaining the size limit set in Step 3, or by explicitly reclustering 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 Programming Guide*.

6.7 How to Shrink the Repository

To shrink the repository files requires taking the repository off-line and restoring it from a backup, because the restore method compacts the extent.

Privileges required: SystemControl, GarbageCollection, and FileControl.

To shrink the repository to its minimum size, make a full backup. Then take the repository off-line 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> run
SystemRepository markForCollection
%
```

For further information about this method, see “MarkForCollection” on page 327.

Step 2. Wait for GemStone to complete the garbage collection and reclaim the space. The time required depends on the size of the repository and, in multi-user mode, on the status of other sessions. For details of various page reclamation mechanisms, see “GcGems Specialized to Reclaim Pages” on page 335.

If other users are logged in, the time required depends in part on the status of other sessions. Space will not be reclaimed until all sessions have committed or aborted any transactions concurrent with the `markForCollection`. For further information, see “To Identify Sessions Holding Up Page Reclamation” on page 346.

Step 3. Make a backup of your repository by sending it the message `fullBackupTo:`*fileOrDevice* MBytes:*byteLimit.* You can use an existing backup only if it was made in full transaction logging mode and you have all transaction logs written since the backup.

For example:

```
topaz 1> run
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 279.

Step 4. Take the repository off-line:

```
topaz 1> run
System shutDown
%
```

Step 5. Remove the existing repository extents. Also remove any extent replicates. For example:

```
% cd $GEMSTONE/data
% rm extentNames
```

Step 6. Make a local copy of the distribution extent, then ensure that you have write permission on the extent that you are copying.

```
% cp $GEMSTONE/bin/extent0.dbf tempDirectory
% chmod +w tempDirectory/extent0.dbf
```

Step 7. Now use **copydbf** to install your copy of the distribution repository as the first (primary) extent. For example:

```
% copydbf tempDirectory/extent0.dbf primaryExtentName
```

Use **chmod** to set the extent permission to what you ordinarily use for your repository.

Step 8. To put the repository back online, issue the **startstone** command:

```
% startstone gemStoneName
```

If you do not specify *gemStoneName*, **startstone** defaults to `gemserver66`.

Step 9. 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 10. 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. For example:

```
topaz 1> run
SystemRepository restoreFromBackup: '/users/bk/dec31'
%
```

This example restores the backup from a single disk file. If you need to restore multiple files or multiple tapes, see “To Restore Multiple-File Backups” on page 287.

GemStone reads the backup 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.
```

Step 11. If the repository was in full transaction logging mode (that is, STN_TRAN_FULL_LOGGING was set to True), restore from any current logs and commit the restore. For example:

```
topaz 1> run
SystemRepository restoreFromCurrentLogs
%
topaz 1> run
SystemRepository commitRestore
%
```

Step 12. Enable logins again:

```
topaz 1> run
System resumeLogins
%
```

6.8 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. You can inquire about the amount of fragmentation in the repository by executing the following expression. Typical values of *aPercentage* range from 10 to 25.

`SystemRepository pagesWithPercentFree:`*aPercentage*

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), and
- the number of data pages having at least the specified percentage of free space.

GemStone automatically schedules reclamation of pages with greater than 10% free space as part of its garbage collection activity.

6.9 How to Recover by Using an Extent Replicate

Recovery using a extent replicate restores the repository to the state of the most recent checkpoint. If an extent replicate is available, this approach is faster than restoring the repository from a backup. Use the following procedure:

Step 1. Examine the system log file to determine the name of the failed extent file or files. For instance, this is an example of the log file entry for an extent failure:

```
[17:22:42.838]
    Log message from user = DataCurator

    Repository read failure, pageId = 1347
      filename = !#dbf!/gslocation/data/extent0.dbf
      reason = RepReadPage failure.
```

Step 2. If you have a replicate of the extent that failed, first make a temporary copy of your system configuration file. Edit the DBF_EXTENT_NAMES list in the temporary file, replacing the name of the bad extent file with the name of the extent replicate. For example:

```
DBF_EXTENT_NAMES = extent0.dbf, replicate1.dbf,
extent2.dbf
```

Remove the name of that extent replicate from DBF_REPLICATE_NAMES (leave the commas). GemStone will not start if the same file name appears as both an extent and an extent replicate. The following example has omitted the extent replicate that was substituted into `DBF_EXTENT_NAMES`:

```
DBF_REPLICATE_NAMES = replicate0.dbf, , replicate2.dbf
```

Step 3. Run **pageaudit** to check for page-level problems. Use the **-z** option to invoke your temporary configuration file.

```
% pageaudit -z temporaryConfigFile
```

Step 4. ***If the pageaudit is successful,*** the easiest course (if the file system itself is usable) is to replace the bad extent with a copy of the good extent replicate:

```
% copydbf replicate1 extent1
```

If a good extent replicate does not exist, but you have recent backups, see the section, "How to Restore a GemStone Repository" on page 281.

Step 5. Restart the Stone repository monitor.

6.10 How to Recover After Repair of the File System

In case of a disk failure or a corrupt file system, the system manager must repair the file system 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, check the condition of the system repository with **pageaudit**. (See "How to Audit the Repository" on page 227 for instructions.)

- 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 the section "How to Restore a GemStone Repository" on page 281.)

To Recover When a File System Must Be Restored

If your system administrator intends to restore the file system from a backup device, before that happens it might be worthwhile to copy the repository to another node or to tape. 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:

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 227. You will need to specify the name and path of the copy using a temporary configuration file when doing **pageaudit** so that audit 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 `extent0.dbf` file that was restored from the file system backup. Check whether the backup was made while GemStone was running.

❒ If any changes were being made to the repository during the operating system backup, `extent0.dbf` may be an inconsistent file that cannot be made to work. In that case, you need to restore from a GemStone backup (see “How to Restore a GemStone Repository” on page 281). However, transaction logs from an operating system backup should be usable.

❒ If the operating system backup was done while GemStone was suspended or shut down, continue to the next step.

Step 2. Do a **pageaudit** to check the current (restored) `extent0.dbf` file. (See the section “To Perform a Page Audit” on page 227 for instructions.)

❒ 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 GemStone backups (see “How to Restore a GemStone Repository” on page 281).

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.*

6.11 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, and
- 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.

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 parameter 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 `rtErrFreeSpaceTreshold` 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 an administrator. 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, 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
The Stone can be configured never to terminate sessions by setting STN_DISKFULL_TERMINATION_INTERVAL *to 0, but 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 in an administrative session 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 previously described.

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 parameter. There are a number of things you can do to create more space in an existing extent, or you can create a new extent.

Creating Space in an Existing 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 nonessential files to create enough space for expanding the repository.
- Invoke `Repository>>markForCollection` or `markGcCandidates` to mark any unreferenced objects so the Stone can remove them. (See the discussion on “Invoking Garbage Collection” on page 325 for details.)

Creating a New Extent

You can 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 and Extent Replicates” on page 190.

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, and sends error `rtErrTranlogDirFull` to each administrative login:

```
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.
```

If the transaction log space is full, you have the following options:

- You can free space by taking some existing log files off-line. 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. Add its maximum file size to STN_TRAN_LOG_SIZES. If transaction logs are being replicated, also add another directory to the STN_TRAN_LOG_REPLICATES configuration option. For information on how to make these changes while GemStone is running, see “To Add a Log and Replicate at Run Time” on page 215.

While it is waiting for space to become available, the Stone continues to process logins and other requests that do not involve writing to the transaction log. Once space becomes available, a new transaction log is created and ordinary operations resume. Waiting sessions can complete operations that were blocked.

Chapter 7

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 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`). 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. If the Stone starts on an existing repository with unrelated transaction logs using the same prefix, it will start numbering with the next available `fileId`. You can control the filename prefix by setting the STN_TRAN_LOG_PREFIX configuration option.

These logs are written to a list of directories (or raw partitions) specified by the STN_TRAN_LOG_DIRECTORIES configuration option, which is treated as a circular list. Each log is limited to the size set for that directory by STN_TRAN_LOG_SIZES. When one log is full, logging switches to the next directory. (What happens when logs have been created in all directories is discussed in Table 7.1.) Collectively, the log files logically form an almost infinite sequential file with a maximum size of 4×10^6 GBytes.

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	STN_TRAN_FULL_LOGGING =FALSE
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 okay)	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 online 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

Use in Recovery from an Unexpected Shutdown

Between checkpoints, GemStone writes each committed transaction to a transaction log *(*Figure 7.1*)*. Then, in the event of a system crash, GemStone can recover by automatically reapplying transactions from the log to the latest checkpoint (Figure 7.2). Multiple transaction logs may be needed.

You can maintain replicates of transaction logs as an added precaution. If GemStone cannot read the primary log during recovery, it tries to read the replicate.

Use ordinary operating system commands to backup the transaction logs in the file system. To backup a transaction log in 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.

Figure 7.1 System Time Line: Normal Operation

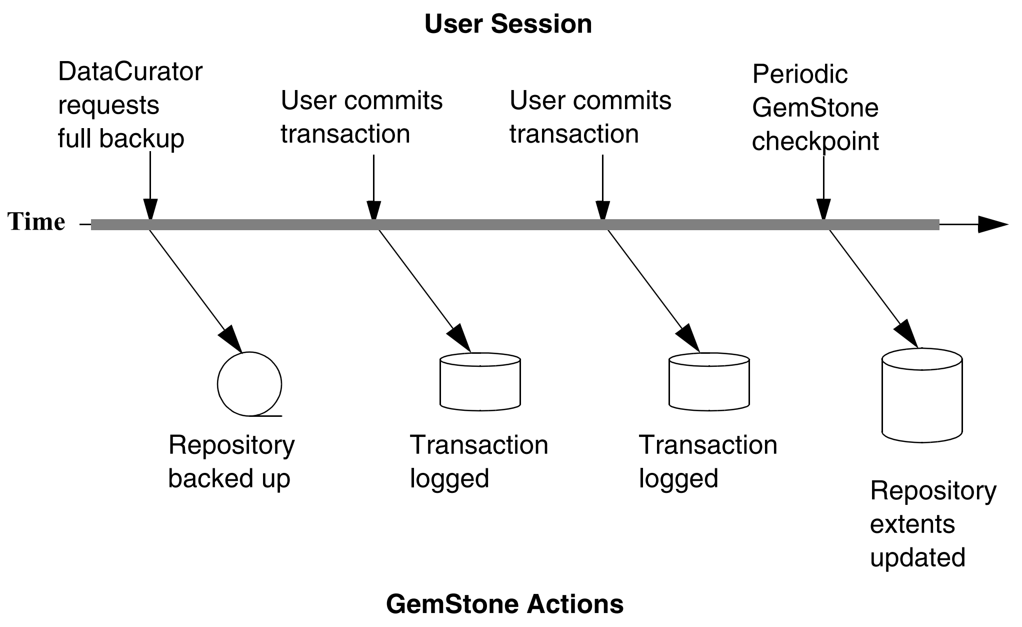
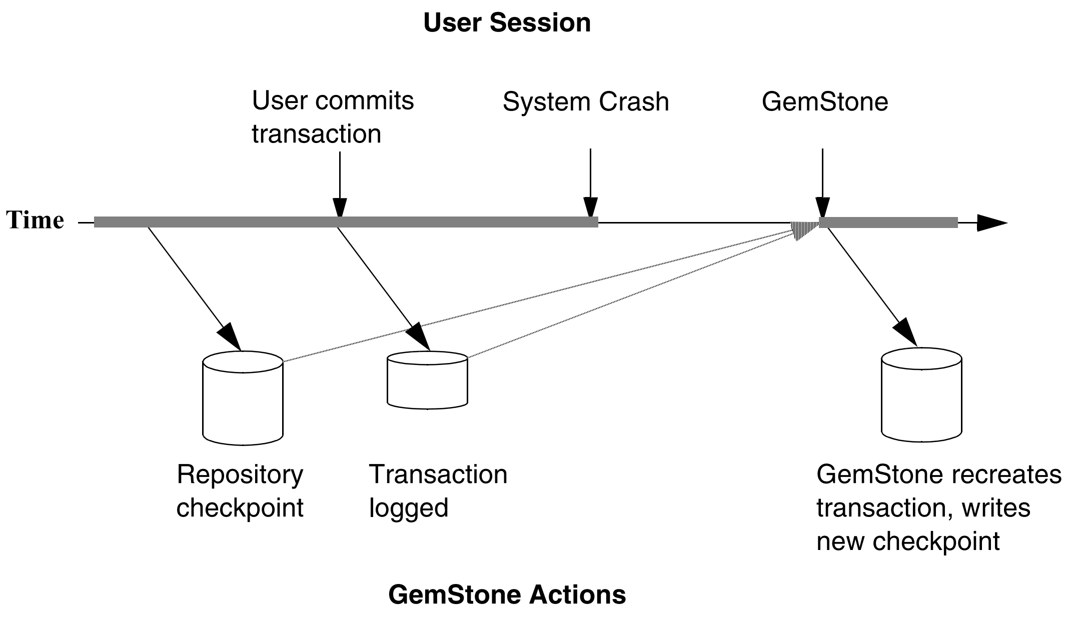


Figure 7.2 System Time Line: System Crash



Use in Rolling Forward from a Backup

An important use of transaction logs is to restore transactions that were committed between the last full backup and a system failure. However, those transactions can be restored only if the repository already is in full transaction logging mode and the backup was made in that mode.

Preconditions

If you have enabled full transaction logging and made a GemStone full backup, you can use the transaction logs to restore transactions committed since the last GemStone 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.

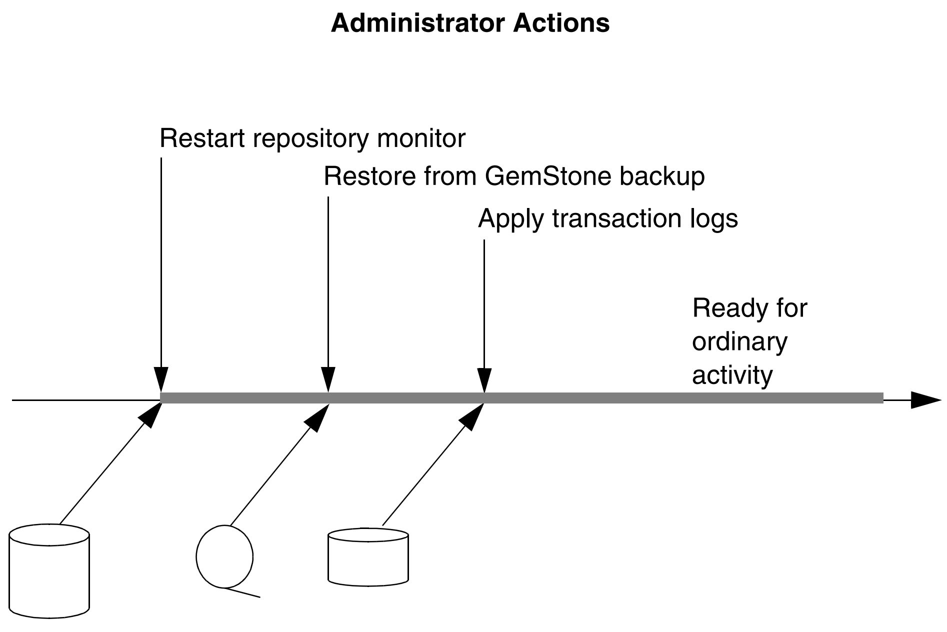
Step 3. Make a GemStone full backup by following the instructions on page 276.

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 in random access format, which serve as a starting point for the restore. Next, you restore the full backup, which loads objects from the backup file. 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. (For the procedure to roll forward from a restored backup, see "B. To Restore Subsequent Transactions" on page 289.)

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 System Time Line: Restoring a GemStone Backup


Administrator Actions
Restart repository monitor
Restore from GemStone backup
Apply transaction logs
Ready for ordinary activity

7.2 How to Manage Full Logging

When the system is operating with the STN_TRAN_FULL_LOGGING configuration option set to True, the 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** 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.

  NOTE
  You may need more than one transaction log to recover, possibly a number of transaction logs, depending on whether there are checkpoints in the transaction logs.

- 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 off-line, although restoring them will take longer.

To Archive Logs

Ordinary operating system 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 a complete log directory at a time in the order listed in the STN_TRAN_LOG_DIRECTORIES configuration option.

NOTE
If you must rename the log files, we recommend that you preserve the

digits in the original file name 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 file name 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 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, see the command descriptions in Appendix B.

You can determine the current size of a transaction log that is in a raw partition by using the method `Repository>>currentTranlogSizeMB`. For information, see Table 7.2.

You can determine oldest transaction log that would be needed to recover from the most recent checkpoint by using the method `Repository>>oldestLogFileIdForRecovery`. This method returns the internal fileId, which is part of the file name 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; be sure that all transaction log files required for recovery are left online.

Similar information can be obtained by applying **copydbf -i** to an extent. For example,

```
% copydbf -i extent0.dbf
Source file: extent0.dbf
  file type: extent  fileId: 0
  Last checkpoint written at: Mon 15 Aug 2011 11:07:54 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 back4.dat
Source file: back4.dat
  file type: backup  fileId: 0
  The previous file last recordId is  -1.
Destination file:  /dev/null
  Full backup started from checkpoint at: Mon 15 Aug 2011 
11:21:20 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/admin/archivelog.sh`. You will need to edit the script to conform to your own partition names and archive location, and then test it.

Compressed transaction logs

Transaction logs are always written in uncompressed format. However, during recover and restore, the Stone repository monitor can read transaction logs that have been compressed using gzip. While compressed transaction logs take up less space, the I/O to these compressed files in much slower, so recovery or restore of compressed transaction logs will take much longer than uncompressed ones.

To Add a Log and Replicate 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*
`replicate:` *replicateSpec* `size:` *aSize*

For example:

```
topaz 1> run
SystemRepository addTransactionLog: '/users/tlogs2'
replicate: '' size: 8
%
```

The argument *replicateSpec* must be consistent with the current list in the STN_REPL_TRAN_LOG_DIRECTORIES configuration option: If that list is empty (logs are not being replicated), *replicateSpec* must be an empty string, as in the preceding example. If STN_REPL_TRAN_LOG_DIRECTORIES is not empty, *replicateSpec* must be a valid device or directory; for example:

```
topaz 1> run
SystemRepository addTransactionLog: '/users/tlogs2'
replicate: '/user3/reptlogs2' size: 8
%
```

The argument *aSize* sets the maximum log size in megabytes for *deviceOrDirectory* (and its replicate). It will be added to the list in STN_TRAN_LOG_SIZES.

You can use the method `System class>>stoneConfigurationAt:` to examine the contents of STN_REPL_TRAN_LOG_DIRECTORIES at run time. For information, see "How to Access the Server Configuration at Run Time" on page 66. 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
`currentLogDirectoryId`	Returns a positive SmallInteger, which is the one-based offset of the current log file into the list of log directory names.
`currentLogFile`	Returns a String containing the name of the transaction log file to which records currently are being appended. If the result is size 0, then a log has failed and a replicate is being used.
`currentLogReplicate`	Returns a String containing the file name of the transaction log file replicate to which records are being appended. The result is a String of size 0 if the current log is not replicated.
`currentTranlogSizeMB`	Returns an Integer that is the size of the currently active transaction log in units of megabytes.

Table 7.2 Repository Methods for Information About Transaction Logs

Method	Description
`logOriginTime`	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.
`oldestLogFileIdForRecovery`	Returns a positive SmallInteger, which is the internal fileId of the oldest transaction log needed to recover from the most recent checkpoint, if the Stone were to crash as of now.

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:

1. starts a checkpoint,
2. waits till the checkpoint completes,
3. starts the new log, and
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> run
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 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. Do a full backup using `Repository>>fullBackupTo:`. See "How to Make a GemStone Backup" on page 276.

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:`. See "How to Restore a GemStone Repository" on page 281.

7.3 How to Manage Partial Logging

Partial logging is GemStone's initial 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

A repository can be changed from partial to full logging simply by changing the STN_TRAN_FULL_LOGGING setting to True and restarting 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.

Chapter

8

Monitoring GemStone

This chapter tells you:

- Where to look for the log files created by GemStone processes
- 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 "Considerations for Large Repositories" on page 148.

8.1 GemStone System Logs

In addition to transaction logs, GemStone creates three types of log files:

- Logs for GemStone server processes (page 222)
- Logs for processes related to individual GemStone gem sessions (page 225)
- Logs for GemStone network server processes, NetLDIs (page 226)

If a GemStone server is running, the **gslist** utility can help you locate its logs. Use **gslist -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, Page Manager, and GcGem 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* ***gslist -c*** *to remove unnecessary files from this locks directory.*

GemStone Server Logs

The Stone repository monitor and its child processes each create a log file in a single location. By default, the log 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 having the default name of `gemserver66`. Log names for child processes also include the process id and a descriptive suffix.

Table 8.1 Representative Log Names for GemStone Server Processes

Typical Name	GemStone Process
`gemserver66.log`	Stone repository monitor
`gemserver66_6936pcmon.log`	Shared page cache monitor
`gemserver66_6966pgsvraio.log`	AIO page server
`gemserver66_6967pgsvrff.log`	Free Frame page server
`gemserver66_6923pagemanager.log`	Page Manager
`gemserver66_6980gc.log` `gemserver66_6988gc*.log`	Garbage collector session Specialized GcGems

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 8.2 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 8.2 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 several useful items:

- the GemStone version,
- the configuration files that were read at startup and 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 (and its processId),
- each #abortErrLostOtRoot sent to a Gem,
- each suspension and resumption of logins, and
- certain changes to the login security system.

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 8.2, it is for processId 6936). Check this log if other messages refer to a shared page cache failure.

When a session logs in from another node and its local shared page cache is enabled, a log is created for the shared page cache monitor on that node. By default, this log is named `startshrpcmon`*PidNode*`.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 (which for remote nodes 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).

AIO Page Server Log

The logs for the repository monitor's AIO page servers are located in the same directory as the Stone's log. These logs are for specific page server processes.

These logs ordinarily are not of interest unless they contain an error message.

Free List Page Server Log

The logs for the repository monitor's free frame page servers are located in the same directory as the Stone's log. These logs are for a specific free frame page server processes.

These logs ordinarily are not of interest unless they contain an error message.

GcGem Logs

Each time the Stone repository monitor starts a new garbage collection (GcGem) session process, a new log is created in the same location as the Stone's log. Each specialized GcGem (page 335) also has its own log. For instance, a new GcGem can be created in response to certain administrative actions that place the repository in single-user mode, such as an object audit. When GcUser logs in again, a new log is opened under a name that includes the new process ID of the GcGem.

These logs show the startup value of the garbage collection parameters that are stored in GcUser's UserGlobals (such as #reclaimMaxPages), and any changes to them.

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.

This log ordinarily is not of interest unless it contains 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 Log names for Session Processes

Typical Name	GemStone Process

`gemnetobject27853node2.log`
`gem12a8-0node2.log (Windows)`
	Gem session process on node2 (serves an RPC application)

`pgsvrmain27819node2.log`	Page server on node2 that the repository monitor uses to create and access its shared page cache on node2
`startshrpcmon27820node2.log`	Shared page cache monitor on node2
`pgsvrmain12397node1.log`	Page server on node1 that the Gem session process uses to access the repository extents on node1

If a process shuts down normally, the log file is 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 107).

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.

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 releases, these directories can be in `/usr`.) This location and name can be overridden by the option **`-l`***logname*e 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:

```
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	Guest mode is not in use (**-g**), but authentication is not required for all NetLDI services (**-s**).
Line 2	Services are not restricted to those listed in `$GEMSTONE/sys/services.dat` (**-n**).
Line 3	Captive accounts are not in use (**-a***name*).

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* typically is invoked to ensure page-level consistency after some kind of system failure, such as a read-write error or a page allocation error. In these cases, a successful page audit indicates that the problem did not affect the committed repository. GemStone must be halted when you perform a page audit.
- An *object audit* checks the consistency of the repository at the object level 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.

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.

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.

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] [-f] [-d] [-h]
```

where:

- *gemStoneName* is the name of the GemStone repository monitor,
- *systemConfig* is the system configuration file, and
- *exeConfig* is the executable configuration file.
- -f indicates to continue running after encountering the first error
- -d specifies to omit audit of data pages, and only audit object table, bitmap, and other system pages.

All arguments are optional in a standard GemStone configuration. If these options are not supplied, **pageaudit** uses `gemserver66` for *gemStoneName*.

- For more information about the **pageaudit** command, see Appendix B.
- For online documentation, type **`pageaudit -/h`** or **`man pageaudit`**.

As **pageaudit** runs, it prints repository statistics to the screen. For example:

```
PAGE AUDIT STATISTICS  mozart sun4u (Solaris 2.9 Generic_117171-08) -
8/15/11 11:12:58 PDT

8192 bytes = 1 GemStone Page = 1 disk blocks
1048576 bytes = 1 Mbytes
Repository Size                          192 Mbytes
Data Pages                                18 Mbytes
Meta Information Pages                    4 Mbytes
Shadow Pages                              0 Mbytes
Free Space in Repository                  168 Mbytes
**** Number of differences found in page allocation = 0.

[11:12:58.986]
   Page Audit of Repository completed successfully.
```

The report contains the following statistics:

Repository Size	The total physical size; the same size that the operating system reports for an extent file.
Data Pages	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	Computed as the number of free pages times the size of a page (8 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 a GemStone Repository” on page 281.)

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 methods abort the current transaction, and the garbage collector session is shut down for their duration. If you have uncommitted changes, an error will be returned and the audit will not run. You will need to manually commit or abort your changes before reattempting the audit.

To have the highest degree of confidence in the audit results, perform the object audit in single-user mode and specify *full checks*. 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* `fullChecks:` *fullChecking*

where *sizeLimit* is object size cutoff (bytes or Oops) below which statistics are not reported, and *fullChecking* is a Boolean indicating whether an error should be returned if the conditions for a detailed consistency check are not met.

When a detailed check is not required, the object audit can be performed under less stringent conditions (other users are logged in or page reclamation could not be completed), but the degree of confidence in the results is reduced because less checking is possible. For example, object does not exist errors may not be detected when object audit is not done in single user mode.

Three 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 performs more general checks. The audit output indicates whether you are performing full checks.
>
> Same as `auditWithLimit: 100000 fullChecks: false`

`Repository>>objectAuditFullChecks`

> `objectAuditFullChecks` is like `objectAudit` except it guarantees that the audit will either run using the more detailed checks or will not run at all.
>
> Same as `auditWithLimit: 100000 fullChecks: 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 performs more general checks. The audit output indicates whether you are performing full checks.
>
> Same as `auditWithLimit:` *sizeLimit* `fullChecks: false`

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 233. If errors are detected, GemStone ordinarily re-scans the repository to provide detailed information.

The optimized methods `Repository`>>`quickObjectAuditLevel1` and `quickObjectAuditLevel2` are intended for use with large repositories, where they run substantially faster than `objectAudit` and `auditWithLimit:`. However, they must be run in single-user mode, and the 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.

`quickObjectAuditLevel1` 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.

`quickObjectAuditLevel2` performs the same integrity audits as `Repository>>quickObjectAuditLevel1`. In addition, all object table entries are audited to verify the disk address of each object. This method will take longer to complete than `Repository>>quickObjectAuditLevel1`.

To perform an object audit:

Step 1. Log in to GemStone using linked Topaz (**topaz -l**).

Step 2. Optionally, put the system in single-user mode (see "How to Enter Single-User Mode" on page 188).

Step 3. Send one of the audit messages to the repository. For example:

```
topaz 1> run
SystemRepository objectAudit
%
```

If errors are reported, and there are unreclaimed dead objects in the repository, there is a chance that the audit errors are in these unreclaimed dead objects, and not in persistent data. You can either proceed with the repair, or re-run the audit after making sure that all unreferenced objects are reclaimed. To do this:

Step 1. Make sure that either the generic GcGem (#GC) or the new specialized EpochGem (#EPC) is running to complete processing of dead objects.

Step 2. Run `markForCollection` (see page 327):

```
topaz 1> run
SystemRepository markForCollectionWait: -1
%
```

Step 3. Run reclaimAll (see page 344):

```
topaz 1> run
SystemRepository reclaimAll
%
```

Step 4. Re-run the object audit.

Audit Errors

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 reference to a nonexistent class or segment.
- 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:

- OopNil is substituted for an invalid object reference.
- DataCuratorSegment is substituted for an invalid segment 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. If the object has a dependency tag, OopNil is stored in the tag to dereference the dependency list.
- 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 GcUser). See "How to Enter Single-User Mode" on page 188. 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> run
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.

Repair Using Backup and Restore

When you create a GemStone backup, all persistent data in the repository is written to the backup file(s). However, some internal structures, such as the list of free OOPs, are not written to the backup file. These structures are rebuilt during the restore from backup process. If corruption is detected in one of the internal structures, making a backup of the repository and restoring it may repair the problem.

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. Classes an application defines will have identifiers of 5277 or higher.

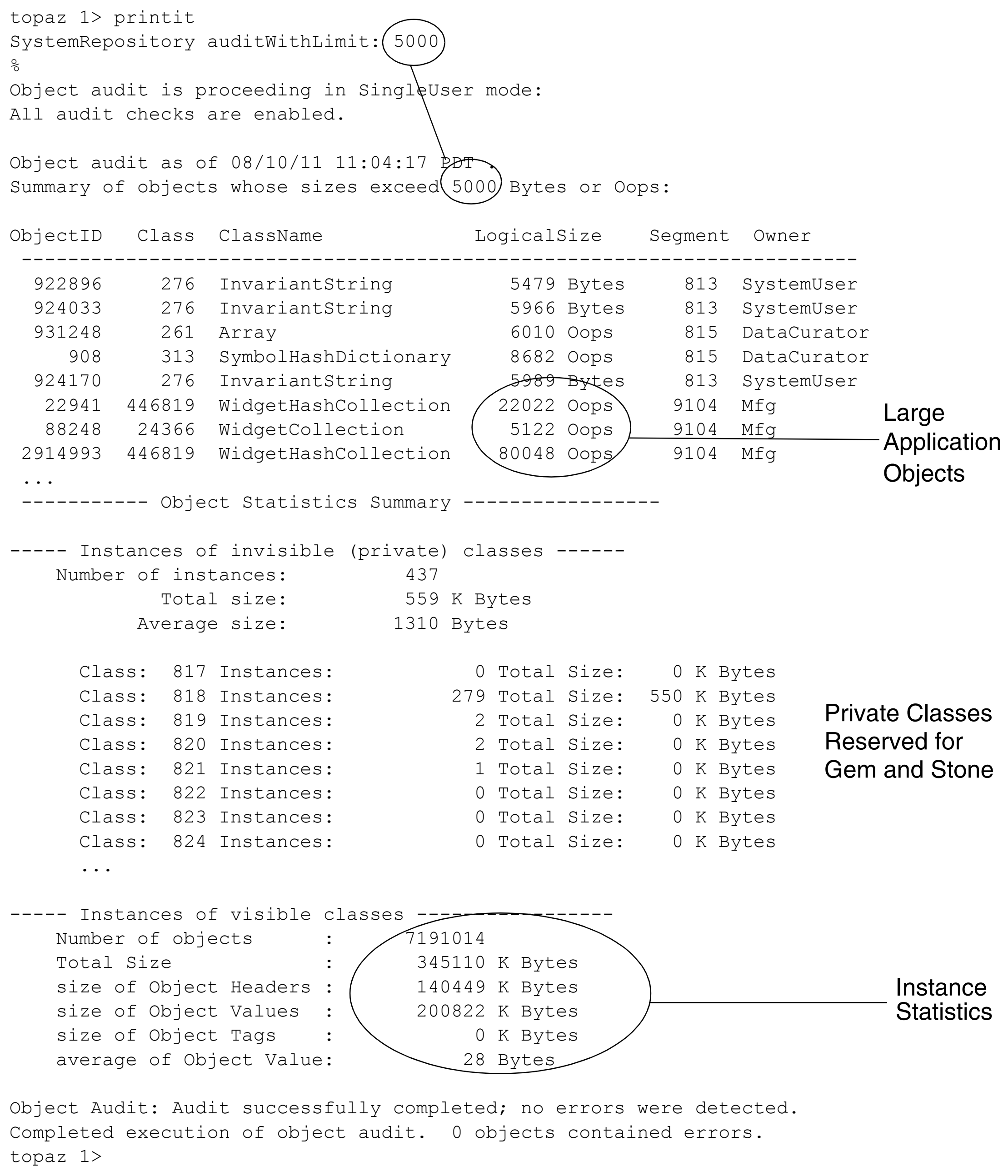
2. Statistics about invisible (private) classes that are reserved for GemStone's use. The number of these classes varies from release to release, and some may not be used in a particular release.

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 47). In this example, the objects occupy an average of 28 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
%
Object audit is proceeding in SingleUser mode:
All audit checks are enabled.

Object audit as of 08/10/11 11:04:17 PDT .
Summary of objects whose sizes exceed 5000 Bytes or Oops:

ObjectID   Class  ClassName             LogicalSize    Segment  Owner
 ----------------------------------------------------------------------
  922896     276  InvariantString          5479 Bytes     813  SystemUser
  924033     276  InvariantString          5966 Bytes     813  SystemUser
  931248     261  Array                    6010 Oops      815  DataCurator
     908     313  SymbolHashDictionary     8682 Oops      815  DataCurator
  924170     276  InvariantString          5989 Bytes     813  SystemUser
   22941  446819  WidgetHashCollection    22022 Oops     9104  Mfg
   88248   24366  WidgetCollection         5122 Oops     9104  Mfg
 2914993  446819  WidgetHashCollection    80048 Oops     9104  Mfg
 ...
 ----------- Object Statistics Summary -----------------

----- Instances of invisible (private) classes ------
    Number of instances:          437
             Total size:          559 K Bytes
           Average size:         1310 Bytes

      Class:  817 Instances:           0 Total Size:    0 K Bytes
      Class:  818 Instances:         279 Total Size:  550 K Bytes
      Class:  819 Instances:           2 Total Size:    0 K Bytes
      Class:  820 Instances:           2 Total Size:    0 K Bytes
      Class:  821 Instances:           1 Total Size:    0 K Bytes
      Class:  822 Instances:           0 Total Size:    0 K Bytes
      Class:  823 Instances:           0 Total Size:    0 K Bytes
      Class:  824 Instances:           0 Total Size:    0 K Bytes
      ...

----- Instances of visible classes -----------------
    Number of objects      :       7191014
    Total Size             :        345110 K Bytes
    size of Object Headers :        140449 K Bytes
    size of Object Values  :        200822 K Bytes
    size of Object Tags    :             0 K Bytes
    average of Object Value:            28 Bytes

Object Audit: Audit successfully completed; no errors were detected.
Completed execution of object audit.  0 objects contained errors.
topaz 1>
```
Large 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.

GemStone includes graphical tools to allow you to record statistics in file and analyze this data graphically. You can also programmatically access these statistics.

A full list of the statistics that are recorded and are available programmatically can be found starting on page 239.

Statmonitor and VSD

GemStone includes the statmonitor utility, which records statistics about GemStone processes to a disk file. You can configure the statistics recorded, how frequently the statistics are collected, and other details.

We recommend running statmonitor at all times, as it provides a valuable record of many aspects of system behavior.

To view this data, VSD (Visual Statistics Display) graphically displays the statistics.

For more details on using statmonitor and VSD, see Appendix G, "statmonitor and VSD Reference", on page 457.

Programmatic Access to Cache Statistics

A set of methods on the System class provide a way for you to analyze performance by programmatically examining the statistics that are collected in the shared page cache. This is the same data that is visible using statmonitor and VSD, although statmonitor and VSD can collect additional OS level information.

A process can only access statistics that are kept in the shared page cache to which it is attached. Sessions that are running on a different node than the Stone use a separate shared cache on that remote node. This means that processes that are on a different node than the Stone, cannot access statistics for the Stone or for other server processes that are attached to the Stone's shared page cache.

Within the shared page cache, GemStone statistics are stored as an array of *process slots*, each of which corresponds to a specific process. Process slot 0 is the shared page cache monitor. On the Stone's shared page cache, process slot 1 is the Stone; on remote caches, slot 1 is the page server for the Stone that started the cache. Subsequent process slots are the page servers, Page Manager, Admin and Reclaim

GcGems, 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.

Name and index of cache statistics

The method `System class >> cacheStatisticsDescription`, when used with display level 1 in Topaz, prints the description of each slot. For example:

```
topaz 1> level 1
topaz 1> run
System cacheStatisticsDescription
%
an Array
 #1 ProcessName
 #2 ProcessId
 #3 SessionId
 #4 LockedPage
 #5 AttachDelta
...
```

Process Types

The specific slots in this array that are used, and in some cases the specific meanings of the values at a specific slot depend on the type of process at that process slot. The process types are:

- Stone
- SPC Monitor
- Page Server
- Gem

The cache statistics descriptions that follow this section include the process type for that statistic; or all if the statistics is meaningful for all process types.

Fetching statistics

`cacheStatistics:` *aProcessSlot* returns an array of statistics for the given process slot.

```
topaz 1> level 1
topaz 1> printit
System cacheStatistics: 1
%
an Array
 #1 Stone
 #2 12256
 #3 0
 #4 -1
 #5 416
...
```

All array elements except the first are Integers. Since not every process type records values for every statistic, the unused elements will have 0 value at that offset.

- `myCacheProcessSlot` returns the process slot in the shared page cache that corresponds to the calling process

  ```
  Topaz 1> printit
  System cacheStatistics: (System myCacheProcessSlot)
  %
  ```

- `cacheSlotForSessionId:` *aSessionId* returns the process slot number for a given session, which must be connected to the same shared page cache as the session invoking the method; if the session cannot be located, the method returns `nil`.
- **`cacheStatisticsForSessionId:`** *aSessionId* returns statistics for a Gem session directly from a session ID, provided that the session is using the same cache as the session invoking this method.

  ```
  Topaz 1> printit
  System cacheStatisticsForSessionId: aSessionId
  %
  ```

Each Gem or page server has a unique number appended to its name, so that data can be related to the correct process in the case where several transient processes successively occupy the same cache slot.

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

GemStone Also 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. These are retrieved using a separate protocol, rather than `cacheStatistics:`.

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:` *anIndex*

Returns the value of the statistic at the designated index (must be in the range 0..47).

`System Class >> _sessionCacheStatAt:` *anIndex* `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:` *aSessionIdt*

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.

Cache Statistics

This section lists the GemStone/S statistics in alphabetical order. The heading indicates the processes for which the statistic is applicable: Stone, Gem, SPC (shared page cache) monitor, Pgsvr (AIO or Free Frame 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.

ActiveProcessCount (SPC monitor)

The number of active processes attached to the shared page cache, as computed by the shared page cache monitor. This value decays slowly. For a faster decay, see RecentActiveProcessCount.

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.

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.

AioWriteFailures (Pgsvr)

Total number of write errors detected by this page server, including write errors that succeeded on retry.

AllSymbolsConflictCount (Gem)

The number of commits that resulted in a conflict on AllSymbols.

AllSymbolsQueueSize (Stone)

The number of sessions waiting for a lock on the global object AllSymbols. Because symbols are canonical (that is, there is only a single instance of each in the system), transactions that create a new symbol must write AllSymbols during commit processing. If the queue typically contains several sessions, you should examine the number of symbols the application creates.

AsyncFlushesInProgress (Pgsvr)

The number of pending extent flush operations.

AsyncWritesCount (Stone)

The number of asynchronous writes performed since the process started (only applies to the Stone repository monitor process).

AsyncWritesInProgress (Stone)

The number of outstanding asynchronous writes queued (only applies to the Stone repository monitor process).

AttachDelta (all)

Calculated by the cache monitor (shrpcmonitor) process and read by other processes. This value controls the number of data pages that the process is allowed to attach in the shared page cache. It is used to control the sharing of the cache resources among multiple processes using the cache.

The attach delta can be either a positive or negative value. If the value is positive, the process may attach that many more pages before it must give up an existing attached page. If the value is negative then the process must give up, that is, release, that many pages before it can attach another page.

AttachDeltaPagesSatisfiedCount (Gem)

The count of pages that the Gem has detached to satisfy the AttachDelta.

AttachedCount (all)

The number of data pages that the process currently has attached in the shared page cache. It indicates how much of a load the application is putting on the cache for resources. See AttachDelta.

BackupHighWaterPage (Stone)

Only used when a full backup is in progress. Highest page ID that full backup has finished writing to the backup file, or INT_MAX (2147483647) if full backup is not active.

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.

BytesCommittedCount (Gem)

The total number of bytes that have been committed by this session.

CacheAttachFactor (all)

An experimental statistic that may be used in a future version of the shared page cache monitor for calculating AttachDelta. If used, it will represent the percentage of free pages that could be attached to this process, based on its CacheMissRatio.

CacheDetachFactor (all)

An experimental statistic that may be used in a future version of the shared page cache monitor for calculating AttachDelta. If used, it will represent the percentage of currently attached pages that this process will have to detach, based on its CacheMissRatio.

CacheEvents (all)

The sum of LocalPageCacheMisses plus LocalPageCacheHits made during the last cycle of the shared page cache monitor loop. This is an experimental statistic that may be used in a future version of the shared page cache monitor for calculating AttachDelta.

CacheMisses (all)

The number of LocalPageCacheMisses made during the last cycle of the shared page cache monitor loop. This is an experimental statistic that may be used in a future version of the shared page cache monitor for calculating AttachDelta.

CacheMissRatio (all)

The ratio of CacheMisses to CacheEvents made during the last cycle of the shared page cache monitor loop, expressed in terms of CacheMisses per 1000 CacheEvents. This is an experimental statistic that may be used in a future version of the shared page cache monitor for calculating AttachDelta.

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. 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. If partial logging is in use, a rapidly increasing CheckpointCount indicates that STN_TRAN_LOG_LIMIT may be set too small.

CheckpointState (Stone)

If if this statistic is 0, it means no checkpoint is active. Values of 1-20 indicate internal stages of a checkpoint.

ClassCacheCount (Gem)

Obsolete; always returns zero.

ClientAborts (Pgsvr)

Number of times the client gem of this page server aborted a transaction.

ClientCallsToStone (Pgsvr)

Approximate number of times the client gem of this page server has made a call to stone. Some calls to stone during the login sequence are not included in this statistic.

ClientCommits(Pgsvr)

Number of times the client gem of this page server committed a transaction.

ClientFailedCommits (Pgsvr)

Number of times the client gem of this page server failed to commit a transaction because of concurrency conflicts.

ClientLostOtRoots (Pgsvr)

Number of times the client gem of this page server was sent a SigLostOtRoot.

ClientPageReads (Pgsvr)

Indicates 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)

Indicates 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.

ClientSigAborts (Pgsvr)

Number of times the client gem of this page server was sent a SigAbort.

ClientStopSessionRequests (Pgsvr)

Number of times the stone has requested this session to stop. Only updated for gems that running on the same host as the stone.

CodeCacheEntries (Gem)

The number of entries in the code cache.

CodeCacheScavengesCount (Gem)

The number of garbage collections of the code cache.

CodeCacheSizeBytes (Gem)

The approximate size in bytes of the code cache. This cache is the private heap memory of the Gem. The code cache contains copies of methods that have been or are being executed by the Gem.

CodeCacheStaleEntries (Gem)

The number of stale methods in the code cache.

CommitCount (Gem)

The number of commits executed by a Gem process (or application linked to a Gem) since the Gem was most recently started.

CommitQueueSize (Stone)

The number of Gem session processes waiting for the commit token.

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 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.

CommitRecordsDisposedCount (Stone)

The total number of commit records disposed by the Stone.

CommitRetryFailureCount (Gem)

The number of commits that failed after exceeding the retry count.

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.

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.

DeadNotReclaimedSize (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. Values greater than about 2000 are an approximation using increments of 65280.

DeadObjsCount (Gem)

The number of dead objects that have been garbage collected by the in-memory garbage collection of temporary objects since the process started.

DeadObjsCount (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.

DetachAllPagesCount (Gem)

The number of times the Gem has detached all pages in the shared cache.

DirtyPageSweepCount (Pgsvr)

The number of times the page server has swept the cache for dirty pages or frames to add to the free list.

EpochGcCount (Stone)

The number of times that the epoch garbage collection process was run by the GcGem since the Stone repository monitor was last started. For a system in steady state, look for uniform periods between runs or a uniform run rate.

EpochNewObjsSize (Stone)

The number of number of new objects created during the last epoch.

EpochPossibleDeadSize (Stone)

The number of possibly dead objects found by the last epoch garbage collection.

EpochScannedObjs (Stone)

The number of objects scanned by the last epoch garbage collection.

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 garbage collected by any of the garbage collection processes (that is, by a Gem's in-memory collection of temporary objects or by epoch garbage collection). 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 becomes greater than 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.

FailedAioCount (Stone)

The number of failed asynchronous I/O operations since the Stone was started.

FailedCommitCount (Gem)

The number of attempts to commit that failed due to concurrency conflicts.

FramesAddedToFreeList (all)

The number of frames added to the free list since the session (for Gems), shared page cache, or Stone started.

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. While scanning for free frames under those conditions is desirable from a system perspective, it represents additional overhead for the particular session.

FramesFromFreeList (all)

The number of times the process acquired a page frame in the shared page cache from the list of free frames.

FreeFrameCount (SPC monitor)

The number of unused page frames in the shared page cache. It 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.

FreeOopCount (Stone)

The number of free OOPs in the free list that have not been allocated to a Gem *and* committed.

FreePages (Stone, Gem)

The size of the free page pool for the repository. Free space in the repository is calculated at 8 KB for each page in the free pool.

GcDeferEpochThreshold (Stone)

The value of the GcUser parameter `#deferEpochReclaimThreshold` that the GcGem is currently using.

GcEpochState (Stone)

The state of the Epoch GcGem. In a single-GcGem configuration, see GcReclaimState. Possible values are:

0 = Not active / sleeping

1 = Setup / background activity

2 = (Not used in Epoch GcGem)

3 = Rebuilding AllSymbols table

4 = Write set union sweep

5 = Epoch GC

GcForceEpoch (Stone)

GcForceEpoch becomes 1 when a user executes `System class>>forceEpochGC` to request that an Epoch GC be started. When the Epoch GC starts, the value returns to 0.

GcHighWaterPage (Stone)

Only used when markGcCandidates is in progress. Highest page ID that markGcCandidates has finished scanning, or INT_MAX (2147483647) if markGcCandidates is not active.

GcInReclaimAll (Stone)

1 if the system is reclaiming pages by executing `System reclaimAll`; 0 otherwise.

GciRpcKeepAlivePacketCount (Gem)

Number of keep-alive packets received from the GCI client on a remote host.

GcLockSession (Stone)

The session ID holding the GC lock or 0 if the GC lock is free.

GcNotConnectedCount (Gem)

The number of times the notConnected objects were garbage collected since the process started.

GcNotConnectedDeadCommittedCount (Gem)

The number of dead objects found during the garbage collection of the notConnected objects that were previously committed.

GcNotConnectedDeadCount (Gem)

The number of dead objects found during the garbage collection of the notConnected objects.

GcPagesNeedReclaiming (Stone)

The number of pages that need reclaiming. This value controls whether promotion of additional objects or epoch garbage collection will be deferred. If this value is greater than GcDeferEpochThreshold (page 248), then epochs are deferred.

GcPossibleDeadSize (Stone)

The number of possible dead objects remaining. Initially, it is the size of the possible dead set received by the GcGem from the Stone. Each time a group of objects is promoted, this value is decremented by the size of that group.

GcPossibleDeadWSUnionSize (Stone)

The size of the possible dead write set union if a sweep is in progress. It is zero if a sweep is not in progress. Because ProgressCount (page 263) during a GcGem sweep may reach this value as its maximum, this value can be used to estimate when the sweep will complete.

GcReclaimMaxPages (Stone)

The value of the GcUser parameter `#reclaimMaxPages` that the GcGem is currently using.

GcReclaimNewDataPagesCount (Stone)

The number of new data pages that the GcGem had to allocate during reclaims since the Stone process was started.

GcReclaimState (Stone)

The state of the normal GcGem (or the Reclaim GcGem in a dual GcGem configuration). Possible values are:

- 0 = Not active / sleeping
- 1 = Setup / background activity
- 2 = Reclaiming shadowed pages
- 3 = Rebuilding AllSymbols table (not used in Reclaim GcGem)
- 4 = Write set union sweep (not used in Reclaim GcGem)
- 5 = Epoch GC (not used in Reclaim GcGem)

GcSweepCount (Stone)

The total number of sweeps of the possible dead write set union that have been done by the GcGem since it started.

GcVoteUnderway (Stone)

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, possible dead write set union sweep pending
- 3 – Possible dead write set union sweep in progress
- 4 – Possible dead write set union sweep completed

Note that all of these phases must complete before the PossibleDead objects can be "promoted" to DeadNotReclaimed objects. The 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 309.

GemsInCacheCount (SPC monitor)

The total number of Gems using the shared page cache whose process slot you are viewing—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).

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.

InTransaction (Gem)

Indicates whether the Gem process is in a transaction.

IntSendCount (Gem)

Obsolete; always returns zero.

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.

LocalCacheFreeFrameCount (all)

The number of frames in the private page cache that are free.

LocalCacheOverflowCount (all)

The number of times a page was moved from private cache to the shared cache because the private cache was full.

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.

LockedPage (all)

Obsolete; always returns zero.

LockReqQueueSize (Stone)

The number of Gem session processes waiting for a commit to complete so that their lock request can be serviced.

LoginWaitQueueSize (Stone)

The number of sessions waiting for login completion.

LogIOSlotCount (Stone)

The number of slots available for asynchronous I/O operations for transaction log writes. If this value drops below 3, the Stone may have to wait for earlier asynchronous writes to complete before starting a new one. This wait time is reported in TimeInLogIOWait (page 269).

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.

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.)

MakeRoomInOldSpaceCount (Gem)

The number of times the oldest temporary object generation filled up. Large values are OK for a large data load session; otherwise, the GEM_TEMPOBJ_CACHE_SIZE configuration option may be too small. We suggest comparing this statistic with NotConnectedObjSetSize (page 256) to see if both are growing. Another useful comparison is that MakeRoomInOldSpaceCount should be less than one-third the size of ScavengeCount (page 265); larger ratios indicate that the cache is too small.

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.

MessagesToStnRebuildScavPagesCommit (Gem)

The number of messages the Gem sent to the Stone while the Gem was rebuilding its list of scavengable pages while processing a 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.

MethodCacheCount (Gem)

Obsolete; always returns zero.

MilliSecPerIoSample (Stone)

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 5 I/Os, MilliSecPerIoSample is computed as 1000/(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. For information about setting this value, see the discussion on page 90. If you want to calculate the current I/O limit, it is given by 1000/(MilliSecPerIoSample * 5).

NewObjsCommitted (Gem)

The number of new objects committed by the most recent transaction committed by this process.

NewSymbolsCount (Gem)

The number of new symbols created by this session.

NonSharedAttached (all)

The number of pages attached by a process that no other process has attached.

NoRollbackSetSize (Gem)

The number of objects in the NoRollbackSet. The NoRollbackSet is used to provide different abort behavior for committed objects. Normal behavior for committed objects is that their state is "rolled back," that is, the modifications to the object made by the transaction are rolled back (removed) by the abort. Objects in the NoRollBackSet do not have this behavior for aborts. Instead, the state of the object is preserved across the abort. This is the kind of behavior desired for “temporary” objects even if they happen to get committed.

Objects are not automatically added or removed from the set by the system. Instead the application has sole responsibility for adding objects to and removing objects from the NoRollbackSet. The contents of the NoRollbackSet can be examined or modified using hidden set methods defined in class System. Although there are no known limits on this set, it is probably best to keep the size under 2K objects.

NotConnectedObjsSetSize (Gem)

The number of objects in the notConnectedObjsSet. This set is used to provide abort behavior for temporary objects written to the disk, that is, for objects that have not been connected to one of the permanent root objects in the repository. (Root objects are the kernel classes and predefined objects like Globals, AllUsers, and so forth.) A large value sometimes indicates that the GEM_TEMPOBJ_CACHE_SIZE configuration parameter set is too small.

This set is updated during commit processing to remove objects that have become connected to permanent objects by the commit. New objects are added to this set when objects move to disk because GEM_TEMPOBJ_CACHE_SIZE overflowed or because an object already on disk references an object in GEM_TEMPOBJ_CACHE_SIZE at commit. The contents of the NotConnectedObjsSet can be examined using the hidden set methods defined in class System. There are a couple of implications for performance tuning. First, if the size of the set is monotonically increasing, it is an indication of garbage objects leaking out of the temporary object space to disk. Second, like the ExportSet, the performance of the system is improved if the size of the set is under 2K objects.

NotifyQueueSize (Stone)

The number of Gem session processes using notifiers.

NumInPgsvrWaitQueue (Stone)

The number of remote sessions logged out and waiting for their pgsvr process to die.

NumInRemoteKillQueue (Stone)

The number of sessions that the page manager is in the process of killing.

NumInRemotePidQueryQueue (Stone)

The number of sessions for which the page manager is trying to determine existance.

NumInSetLostOtQueue (Stone)

The number of sessions waiting for set lostOt in shared caches.

NumSlotsPendingReuse (SPC Monitor)

Number of slots in the shared page cache that will be reused when the stone completes cleanup of a session which previously used the slot.

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 the oldest commit record is not referenced by any session.

OldestCrSessNotInTrans (Stone)

The session ID of the oldest session that is not in a transaction that is currently referencing the oldest commit record. This session may be preventing the commit record from being disposed.

OldSpaceOverflowCount (Gem)

The number of times objects were moved from old space into the NotConnectedSet because old space filled.

OopNumHighWaterMark (Stone)

The highest number of object identifiers allocated by the system.

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.

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.

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.

PageKindsWrittenByGems (SPC monitor) PageKindsWrittenByStone (SPC monitor)

Each of these statistics consists of an array of counts for 21 page kinds. Together, these arrays show the current contents of the shared page cache in terms of the kind of page and who initiated it (a Gem or the Stone). Table 8.3 shows the page kinds that may be of interest to users. The statistics are available only for the shared page monitor's slot (currently Slot 0).

Table 8.3 Page Kinds in Shared Page Cache

Index	Page Kind Code	Page Kind
1	0	Invalid (empty) page
2	1	Root page
3	2	(for internal use)
4	3	Old commit record page
5	4	Data page
6	5	Object Table internal page

Table 8.3 Page Kinds in Shared Page Cache (Continued)

Index	Page Kind Code	Page Kind
7	6	Object Table leaf page
8	7	Bitmap internal page
9	8	Bitmap leaf page
10–12	9–11	(for internal use)
13	12	Bitlist page (a form of bit array)
14–19	13–18	(for internal use)
20	19	Lost Object Table page
21	20	Commit record page

PageMgrPagesNotRemovedFromCachesCount (Stone)

The total number of pages 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.

PageMgrPagesRemovedFromCachesCount (Stone)

The total number of pages the Page Manager has successfully removed from all shared page caches.

PageMgrPagesReceivedFromStoneCount (Stone)

The total number of pages the Page Manager session received from the Stone to remove from shared page caches.

PageMgrRemoveFromCachesCount (Stone)

The total number of pages the Page Manager has successfully removed from all shared page caches.

PageMgrRemoveFromCachesPageCount (Stone)

The total number of pages 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 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 always 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.

PageReadsRebuildScavPagesCommit (Gem)

The total of pages read while the Gem was rebuilding its list of scavengable pages while processing a 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.

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.

PagesNeedRemovingThreshold (Stone)

The threshold for the Page Manager to process the backlog described by PagesWaitingForRemovalInStoneCount.

PagesNotFoundInCacheCount (SPC monitor)

The total number of pages not found in the shared cache when the Page Manager Gem or cache page server attempted to remove them.

PagesNotRemovedFromCacheCount (SPC monitor)

The number of pages that the cache page server or Page Manager Gem was unable to remove from the cache. Requests to remove pages come from the Stone.

PagesRemovedDirtyFromCacheCount (SPC monitor)

The number of dirty pages successfully removed from the cache by the cache page server or the Page Manager at the Stone's request.

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.

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.

PasswordPagesWrittenByGem (SPC Monitor)

Number of password pages written by a gem.

PasswordPagesWrittenByStone (SPC Monitor)

Number of password pages written by the stone.

PersistentPagesDisposed (Stone)

The number of persistent pages (pages already checkpointed) that have been disposed of while in the Stone's private cache.

PgsvrPid (Gem)

The Process ID of the session's page server (remote Gems only).

PinnedPagesCount (all)

The number of pages that the process has pinned (locked) in the shared cache. Pages may be pinned by more than one process at the same time.

PossibleDeadSize (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. The object is not declared ("promoted to") dead until each active session verifies the absence of such references during its next commit or abort. Values greater than about 2000 are an approximation using increments of 65280.

PrivateAttachLimit (all)

An intermediate value used to calculate the AttachDelta.

PrivatePinnedPagesCount (all)

The number of pages that the process has pinned (locked) in its private page cache.

ProcessesWaitingForQueueLocks (SPC Monitor)

Number of processes attached to the shared cache which are spinning while attempting to acquire a queue lock.

ProcessId (all)

The operating system processId for the process associated with this shared page cache process slot.

ProcessName (all)

Astring that identifies the process kind (Gem, Stone, page server, or shared page cache monitor).

ProgressCount (Gem)

Can be used to monitor the progress of certain Repository methods that may run for extended periods.

During `markForCollection`, ProgressCount for that Gem's cache slot first indicates the number of objects swept during the mark-sweep (transitive closure) phase. The transition to the second phase is marked by ProgressCount being reset to zero. During the second phase, it indicates the number of possible dead objects identified by taking the difference between the universe and those objects found during the mark-sweep phase.

During `fullBackupTo:` and `restoreFromBackup:`, ProgressCount for that Gem's cache clot is the number of objects written to or restored from the backup file.

During `objectAudit` or `auditWithLimit:`, Progress count for that Gem's cache slot is the number of objects audited.

For the GcGem's cache slot, ProgressCount during the reclaim task is the number of pages reclaimed. During epoch garbage collection, ProgressCount first is the number of objects swept, and then is number of objects identified as possibly dead (the same as during `markForCollection`, above).

RcConflictCount (Gem)

The number of commits that resulted in an reduced-conflict (RC) conflict.

RebuildScavPagesForCommitCount (Gem)

The total number of times the Gem rebuilt is list of scavengable pages while processing a commit.

RecentActiveProcessCount (SPC monitor)

The number of active processes attached to the shared page cache. It is computed more often than ActiveProcessCount and has decays quickly.

ReclaimCount (Stone)

The number of reclaims performed by a GcGem process since the Stone repository monitor was last started.

ReclaimedPagesCount (Stone)

The number of pages reclaimed by a GcGem reclaim 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.

ReclaimWaitQueueSize (Stone)

The size of the queue for session that are waiting for `reclaimAll` to complete.

RecoverCrBacklog (Stone)

The size of the commit record backlog that was in effect during the generation of the tranlog record currently being replayed during system recovery or restore.

RecoverTranlogBlockId (Stone)

The block ID of the tranlog currently being replayed during system recovery or restore.

RecoverTranlogFileId (Stone)

The file ID of the tranlog currently being replayed during system recovery or restore.

RemoteCachesNeedServiceCount (Stone)

The number of outstanding requests to start or stop a remote shared page cache. Requests are initiated by the Stone and executed by the Page Manager session.

RemoteSessionCount (Stone)

The number of sessions that are running on a host other than the Stone's host.

RemoteSharedPageCacheCount (Stone)

The total number of remote shared page caches attached to the system.

RemoteSharedPageCacheMax (Stone)

The maximum number of remote shared page caches that may be used with this system.

RunQueueSize (Stone)

The number of Gem session processes waiting for service from the Stone repository monitor.

ScavengeCount (Gem)

The number of times that the in-memory temporary object garbage collector was executed since the process started.

SessionId (all)

The GemStone sessionId associated with this client.

SessionNotVoted (Stone)

The Stone sessionId of a session that has not yet voted on the possible dead objects.

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.

SharedAttached (SPC monitor)

The number of data pages in the shared page cache that are attached by more than one process. A large value indicates that processes may be accessing the same objects, while a small value indicates that processes are mostly accessing different objects. This value is valid (non-zero) only for the shared page cache monitor process slot.

SigAbortCount (Gem)

The number of abort signals that have been sent by the Stone to this session. This counter is incremented when the Stone sends the signal, even if the session ignores it.

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.)

SigLostOtCount (Gem)

The number of lost object table signals that have been sent by the Stone to this session.

SmcQueuesInUse (Stone)

Number of shared memory queues in use on this shared page cache. On stone's cache, this value can be 1, 2, 4, or 8. It is always zero on remote shared page caches.

SmcQueueSize0...7 (Stone)

The number of sessions in the given SMC (shared memory communication) waiting to be added to the run queue by Stone. N is

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.

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.

StoredLomObjsMapSize (Gem)

Number of objects in the stored LOM objects map.

StoredPomObjsMapSize (Gem)

Number of objects in the stored POM objects map.

StnGetLocksCount (Stone)

The total number of times the Stone retrieved the lockset and passed it to a remote Gem.

StnLoopCount (Stone)

The total number 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.

StnLoopNetPollCount (Stone)

The total number of times the NetPoll() function was called from the Stone's main control loop.

StnLoopNetPollOobCount (Stone)

The total number of times the NetPollOob function was called from the Stone's main control loop. This function is used to poll out-of-band sockets to Gems for activity.

StnLoopState (Stone)

An integer identifying 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.

StnLoopTimeInNetPoll (Stone)

The total amount of real time (in milliseconds) the Stone spent calling the NetPoll() function.

StnLoopTimeInNetPollOob (Stone)

The total amount of real time (in milliseconds) the Stone has spent in the NetPollOob function.

StopSessionCount (Gem)

Number of times the stone has requested this session to stop. Only updated for gems that running on the same host as the stone.

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.

TempPagesDisposed (Stone)

The number of temporary pages (pages allocated since the last checkpoint) that have been disposed.

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.

TimeInGcNotConnected (Gem)

The cumulative number of milliseconds that the Gem or Stone has spent garbage-collecting the NotConnectedSet since the session started. This statistic also includes time spent performing `makeRoomInOldSpace` and generational scavenge (epoch garbage collection) operations.

TimeInLogIOWait (Stone)

The total amount of real time (in milliseconds) that the Stone has had to wait for prior asynchronous transaction log writes to complete before starting a new one. A high value indicates problems with asynchronous writes on the Stone's machine.

TimeInPgsvrNetReads (Stone, Gem)

The cumulative number of milliseconds that the Gem or Stone has spent reading data across the network from the page server.

TimeInPgsvrNetWrites (Stone, Gem)

The cumulative number of milliseconds that the Gem or Stone has spent writing data across the network to the page server.

TimeInRebuildScavPagesCommit (Gem)

The total amount of real time the Gem spent rebuilding its list of scavengable pages while processing a commit.

TimeInScavenges (Gem)

The CPU time (in milliseconds) spent in the in-memory temporary garbage collector.

TimeInStnGetLocks (Stone)

The total time spent by the Stone retrieving the lockset and passing it to a remote Gem.

TimeInStonePageDisposal (Stone)

The total amount of real time (in milliseconds) that the Stone has spent performing page disposal tasks.

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.

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.

TimeWaitingForStone (Gem)

The total time the Gem spent waiting for a response from the Stone.

TimeInUpdateUnionsCommit (Gem)

The total real time the Gem spent updating its unions while waiting for the commit token.

TotalAborts (Stone)

The number of abort operations performed system-wide since the Stone was started.

TotalAttached (SPC monitor)

The total number of data pages attached in the cache (the sum of AttachedCount for all processes). The results are valid (non-zero) only for shared page cache monitor process slot.

TotalCommits (Stone)

The total number of commits (excluding read-only commits) performed by all processes since the Stone repository monitor was last started.

TotalNewObjsCommitted (Stone)

The total number of new objects committed by all Gems.

TotalObjsCommitted (Gem)

Obsolete; always returns zero.

TotalSessionsCount (Stone)

The total number of sessions currently logged in to the system.

TransactionLevel (Gem)

Describes the state of the Gem.
1 = in a transaction.
0 = outside a transaction.
–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.

VcCacheScavengesCount (Gem)

The number of garbage collections of the VC space. VC space is a memory region private to the virtual machine.

VcCacheSizeBytes (Gem)

The total size in bytes of the VC space. This space is the private heap memory of the Gem.

VoteNotDead (Gem)

The number of objects that the Gem process removed from the possibleDead set the last time that it voted on the possibleDead.

WaitsForOtherReader (all)

PageRead operations avoided by waiting for read already in progress by another process.

Chapter
9
Making and Restoring Backups

This chapter explains how to make backups of your repository and, should it become necessary, how to use them to restore the repository. We recommend that you use backup and restore methods provided as part of the GemStone kernel. However, it is also possible to use operating system backups, if you comply with the precautions this chapter describes. Finally, this chapter also describes how to recover in the case of a disk failure or corrupted file 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.

Use the method `Repository>>fullBackupTo:`, which copies all the objects in the repository and arranges them in a compact form. This backup can be made while the repository is in use. Then, if you have enabled full transaction logging, the logs save information about all objects committed since the backup.

The advantage of this procedure over operating system backups is that you have full control of backups, and the backups can be made while users are logged in. Also, dynamic internal data structures are copied and will be restored, improving performance of such routine maintenance tasks as garbage collection.

In the event of a subsequent repository failure, the last full backup and the transaction logs can restore the current repository. In the absence of both extent replicates and full transaction logging, a media failure can cause the loss of all updates since the last backup.

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 213.

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.

Backups Are Made While GemStone Is Running

GemStone programmatic backups are always made while the Stone repository monitor is running. The method `fullBackupTo:` saves the most recently committed version of the repository in a way that is consistent from a transaction viewpoint. Other sessions can continue to commit transactions, but those transactions will not be included in the backup in progress.

A full backup has 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 "What Is Garbage?" on page 304.)

   If necessary, the I/O rate can be limited for the session performing the backup so it does not monopolize system resources. For further information, see "To Limit the Session I/O Rate" on page 89.

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.

Which Files Can Be Backed Up by the Operating System

While the Stone repository monitor is running, only transaction logs can be backed up safely using operating system facilities. Extents cannot be backed up safely that way because of the manner in which they are written.

WARNING
Do not make operating system backups of extents while the Stone repository monitor is running because they are likely to be unusable. See "Why Operating System Backups May Not Be Usable," below. Always shut down the Stone before making backups using the operating system.

Operating system facilities and **copydbf** can be used safely to backup a repository ONLY following an orderly shutdown, such as by **stopstone** *gemStoneName*. During the shutdown process, a checkpoint is performed in which all committed transactions are written to the extents and to any replicates. 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.

Recovery using backups by the operating system requires a special procedure, which is described under "How to Restore from an Operating System Backup" on page 297.

Why Operating System Backups May Not Be Usable

If changes were being made to the logical repository during an operating system backup, the individual extent files in the backup may form an inconsistent repository that cannot be made to work.

To protect your data, we recommend that you create backups of the repository using the method `Repository>>fullBackupTo:` as described in this chapter, rather than using operating system methods such as **dump**, **tar** or **cp**. The GemStone backups created with `fullBackupTo:` are the most reliable because they are written from a transaction point of view and save a consistent state of the repository. The GemStone backups also have the advantage of being made while the repository is online and are much smaller.

9.2 How to Make a GemStone Backup

Privileges required: FileControl.

The message `fullBackUpTo:` forces a checkpoint of the repository at the time the method is executed and then creates a backup from that checkpoint. Other sessions can continue to commit transactions, but those transactions will not be included in the backup in progress. There are two related messages:

`Repository>>fullBackupTo:`*fileOrDevice* and
`Repository>>fullBackupTo:`*fileOrDevice* `MBytes:`*mByteLimit*

If full transaction logging is enabled, the backup file together with logs created since the backup contain all information necessary to restore the repository to its current committed state.

The argument *fileOrDevice* specifies the file or device where the backup is to be created. Backups can be created 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 278.

WARNING
If fileOrDevice *runs out of space, the backup will terminate with a system I/O error at that point. The backup will be unusable. To avoid having to repeat the entire backup, make sure the device has sufficient space or set* mByteLimit *appropriately.*

The argument *mByteLimit* in the second message lets you create a multiple-file backup by limiting the size of each part. For further information and an example, see "To Create a Backup in Multiple Files" on page 279. If you don't want to limit the size of the backup file, specify a *mByteLimit* of 0 or use the first message, which omits `MBytes:`*mByteLimit* 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.

To perform a full backup, send your repository the message `fullBackupTo:`*fileOrDevice*. For example:

```
topaz 1> run
"Create a full backup of SystemRepository in the file
      '/users/backups/aug13.11'"
SystemRepository fullBackupTo:'/users/backups/aug13.11'
%
true
```

The backup file named `aug13.11` is created.

During the backup, the session is put in 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.`**

The session performing the backup performs an initial abort; if this session has any uncommitted changes, it will return an error and will not run.

If the backup file already exists, the method returns an error.

If the *fileOrDevice* argument is an empty string, an error will be returned. You must specify the name of a file or device, not a directory name.

Additional Performance Tips

For the session performing the backup, the statistic ProgressCount (described on page 263) indicates the number of objects written to the backup file so far. If you know the number of objects in the repository, you can use this to determine how far the backup has progressed.

You can often improve both backup and restore performance if you increase the size of the shared page cache.

You can usually improve both backup and restore performance if you also increase the size of the private page cache for the Gem performing the backup or restore. For example, in that Gem's configuration file, include the line:

```
GEM_PRIVATE_PAGE_CACHE_KB = 65536;
```

Backups and Garbage Collection

NOTE
Tips in this section will be easier to understand if you have first read and

understood the section entitled "Basic Ideas" on page 304 in Chapter 10, "Managing Growth."

Because shadow objects must be backed up, it is more efficient to run a backup when there are few shadow objects. If possible, first check the statistic PagesNeedReclaimSize (page 260). If that statistic is high, run one or more Reclaim GcGems before performing the backup. (Such GcGems are described in "GcGems Specialized to Reclaim Pages" on page 335.) This action could hasten step 2 considerably.

Dead objects waiting to be reclaimed (measured by the statistic DeadNotReclaimedSize, described on page 245) 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 309). 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 statistics PossibleDeadSize (described on page 262) and GcPossibleDeadSize fall, and the statistic DeadNotReclaimedSize (described on page 250) rises.

To Create a Backup on a Remote Node

You can backup to or restore from drives that are NFS-mounted. The following example uses NRS to access a drive located on another node that are not mounted. Performing a backup across the network is likely to take much longer than writing it to a local device.

```
topaz 1> run
SystemRepository
fullBackupTo:'!@flute!/users/gsadmin/bkup4.dat'
%
```

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:`*byteLimit.* 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 megabytes 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> run
"Start a full backup of SystemRepository in the file
      '/users/backups/aug13.11'"
SystemRepository fullBackupTo: '/users/backups/aug13.11' MBytes:
10
%
GemStone has 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.*

Both `fullBackupTo:` *MBytes* and `continueFullBackupTo:`*fileOrDevice* `MBytes:`*mByteLimit* return true if the backup is complete, or "GemStone has finished writing backup file...", if the backup is incomplete and the backup process must be continued.

For example:

```
topaz 1> run
 "Continue a full backup by writing a second file to
      '/users/backups/aug13.11_2'"
SystemRepository continueFullBackupTo:
'/users/backups/aug13.11_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** *fileName* **-i**.

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 a file system that is NFS-mounted.

Backup files written in compressed mode are automatically appended with the suffix `.gz` if that suffix is not specified by the user 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 it accordingly. Even a backup originally created in uncompressed mode, then later compressed externally with gzip, is readable by `restoreFromBackup:`.

NOTE
Tranlogs 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

If you want 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/aug13.11 /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> run
BackupLog
%
fullBackup to /users/backups/aug13.11 started at Aug/15/2011 
11:39
```

9.3 How to Restore a GemStone Repository

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 ordinarily takes place in two phases:

1. Restore the repository from the last GemStone backup file or tape.
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.)

Figure 9.1 illustrates the process.

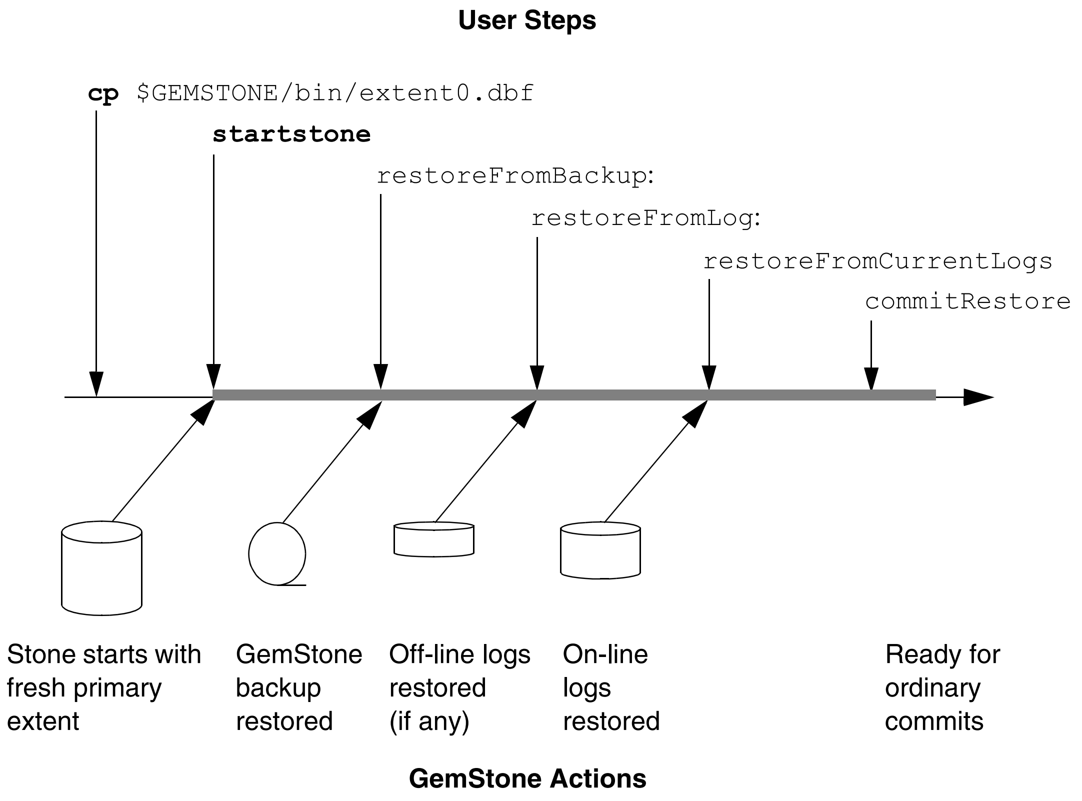
To protect your data, we recommend that you restore from backups created by GemStone, not from backups created by operating system commands such as **dump**, **tar** or **cp**. GemStone backups are created by the method `Repository>>fullBackupTo:`. These backups are the most reliable because they are written from a transaction point of view and save a consistent state of the repository. (If you need to restore backups made by operating system commands, see the section "How to Restore from an Operating System Backup" on page 297.)

NOTE

Backups created by a dump of the whole file system may or may not be usable. Before you consider restoring from OS-level backups, be sure to read "How to Restore from an Operating System Backup" on page 297.

Before you begin, make sure you have a backup of the system repository that is complete, and make sure that the backup was created by GemStone with the method `fullBackupTo:`. (See the discussion "How to Make a GemStone Backup" on page 276.) If full backups of your repository require more than one backup file, restoring backup file 1 without backup file 2 does not give you a usable version of the repository. The objects on backup file 1 are copied, but the transaction cannot be committed until you restore backup file 2. (The Topaz **commit** command does not commit a partial restore, even though the message may say the commit was successful.)

Figure 9.1 System Time Line: Restoring a GemStone Backup



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, the allocation of objects to extents 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.

A. To 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. 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.

Since copydbf requires write permission to the extent you are copying, you will need to make a local copy of `$GEMSTONE/bin/extent0.dbf` to use copydbf.

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 GemStone 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. Make sure that you have full backups of good repository files. If the backup consists of multiple files, the complete set must be available.

Step A3. 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.

CAUTION
Do NOT delete the transaction log files as of the crash – leave them online without moving them.

Step A4. Remove all extent files, and removedbf all extents on raw partitions, that are specified in DBF_EXTENT_NAMES in your configuration file. Similarly, remove all replicates specified in DBF_REPLICATE_NAMES.

Copy the unmodified distribution extent in `$GEMSTONE/bin` to the location of your primary extent, which is the extent listed first in DBF_EXTENT_NAMES (by default `$GEMSTONE/data/extent0.dbf)`. For example:

```
% cp $GEMSTONE/bin/extent0.dbf $GEMSTONE/data/extent0.dbf
```

or

```
% cp $GEMSTONE/bin/extent0.dbf /tempDir/extent0.dbf
% chmod +w /tempDir/extent0.dbf
% copydbf /tempDir/extent0.dbf $GEMSTONE/data/extent0.dbf
```

Step A5. Use **chmod** to give the copy the same permissions you ordinarily assign to your repository files.

```
% chmod 600 $GEMSTONE/data/extent0.dbf
```

Step A6. 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, and a new transaction log sequence, when you start the restoration process (Step A9). This is a independent sequence of transaction logs from the one that you will be restoring, although the names may look similar.

Step A7. Use **startstone** to restart the Stone.

Step A8. Log in to GemStone as DataCurator or SystemUser using linked Topaz (**topaz -l**). Remember that the password will be the original one, 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 A9. Restore the most recent full backup to the new repository by sending the message `restoreFromBackup:` *fileOrDevice*. 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 restoring backups from more than one file, see “To Restore Multiple-File Backups” on page 287.

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 will persist across login sessions and stopping and starting of the Stone repository monitor.

```
topaz 1> run
SystemRepository restoreStatus
%
Restore is not active
topaz 1> run
SystemRepository restoreFromBackup: '/bk/aug13.11'
%
```

If the full backup is contained in one file, the system commits the restore and returns a summary and status. For instance:

```
The restore from full backup completed, with 30569
objects restored and 0 corrupt objects not restored.
```

Step A10. If the backup was made in a repository with the configuration parameters STN_TRAN_FULL_LOGGING set to true, the status line is similar to this:

```
Ready for restore from transaction log(s).
```

Continue with “B. To Restore Subsequent Transactions” on page 289.

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 297.

If full logging was disabled (partial logging was in effect), 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.

Performance Tips

Restoring from backup usually takes 10-30% longer than the full backup took.

For the session performing the restore, the statistic ProgressCount (described on page 263) indicates the number of objects restored from the backup file so far. If you know the number of objects in the backup file, you can use this to determine how far the restore has progressed.

You can improve restore performance by using one aio page server for every extent that resides on its own dedicated disk drive, but only if the repository uses weighted extent allocation mode, and the disks are on separate physical disks. For example, if you're 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 `restoreNoShadowsFromBackups:` methods, rather than restoring backups one by one. These methods require you to specify all backup files in the correct order in an Array.
- Maximize the repository shared cache size, and the private page cache size for the process performing the restore, keeping the total within the 4 GB space that a 32-bit process can address.
- Set the GEM_FREE_FRAME_LIMIT low; for example, a setting of 500.
- Place extents on striped file systems.
- Make sure the backup files being restored are not on the same disk spindles as the extents or tranlogs.

To Restore Multiple-File Backups

If you are restoring a backup that occupies more than one file, each backup file must be restored in the correct 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 full backup.)

Repeat Step A9 on page 285, using the next file in sequence, for each part of the backup. If you are uncertain of the sequence in which to restore a particular file, use **copydbf** *fileName* **-i** to display its fileId.

Another way is to use `restoreFromBackups:` *arrayOfFilesOrDevices* and include all files 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. This example restores a backup that occupies two files by placing the file names in an array:

```
topaz 1> run
SystemRepository restoreFromBackups:
#( '/backups/aug13.11_1'
   '/backups/aug13.11_2' )
%
[restore info]:GemStone is starting to restore the 
repository from 2 backup files.
[restore info]:GemStone is verifying 2 backup files.
[restore info]:GemStone has finished verifying 2 backup 
files. No errors were detected.
[restore info]:GemStone attempting to open 
/backups/aug13.11_1_.
[restore info]:GemStone reading from /backups/aug13.11_1_.
[restore info]:GemStone attempting to open 
/backups/aug13.11_2_.
[restore info]:GemStone reading from /backups/aug13.11_2_.
[restore info]: GemStone has finished restoring 872 data 
pages and 25 object table pages.
[Info]: Logging out session 3 at 8/20/11 13:31:47 PDT
The restore from full backup completed, with 44249 objects 
restored and 0 corrupt objects not restored.

Ready for restore from transaction log(s).
```

Another alternative is to use `restoreNoShadowsFromBackups:` *arrayOfFilesOrDevices*. This method is the same as `restoreFromBackups:` except that partially filled data pages are not added to the list of scavengable 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 286.

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.

B. To 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. (You may need multiple logs, depending on whether there are checkpoints in the transaction logs.) 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 297.

Step B1. Before continuing the restore process, you must log in again. (The `restoreFromBackup:` method (Step A9) 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 `tranlog37.dbf`:

```
topaz 1> run
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to Tue 16 Aug 2011 14:52:06 PDT
next fileId = 37
```

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, tranlog*NN*.dbf. For transaction logs in raw partitions, use **copydbf** *rawPartition* **-i** 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 66).

Step B3. Restore the transaction logs in time sequence, beginning with the log identified by `restoreStatus`. How you do this 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 294.

If only current on-line transaction logs are needed - that is, all transaction log files beginning with `tranlog`*fileId*`.dbf` are in the locations specified by the STN_TRAN_LOG_DIRECTORIES or STN_REPL_TRAN_LOG_DIRECTORIES configuration options, skip to Step B4.

If any of the older transaction logs that are needed have been moved to a different disk location, you will need to restore from archived logs. You can restore the logs individually, or restore all logs in a set of directories.

To restore all archive logs, send the message `Repository>>restoreFromArchiveLogDirectories:` ... `replicatePrefix:`. 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> run
SystemRepository restoreFromArchiveLogDirectories:
  #( 'GS-archive' )
tranlogPrefix: ''
replicateDirectories: #()
replicatePrefix: '' .
%
[Info]: Logging out session 2 at 8/20/11 16:20:21 PDT
Restore from transaction log(s) succeeded.
restoreFromBackup succeeded.
```

See the method comment in the image for details. A directory location can include an NRS for a remote node, but a NetLDI must be running on that node.

If you only have a few transaction logs to restore, you can alternatively execute `Repository>>restoreFromLog:`*fileOrDevice* for each file or raw partition in time sequence. For example:

```
topaz 1> run
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to Tue 16 Aug 2011 17:28:13 PDT
next fileId = 38
topaz 1> run
SystemRepository restoreFromLog: '/GS-archive/tranlog38.dbf'
%
Restore from transaction log(s) succeeded.
```

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 on-line logs.

Step B4. Restore transactions from all remaining on-line 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 or STN_REPL_TRAN_LOG_DIRECTORIES.

```
topaz 1> run
SystemRepository restoreFromCurrentLogs
%
Restore from transaction log(s) succeeded.
```

You may also use `Repository>>restoreFromLog:`*fileOrDevice* to restore individual current transaction logs.

You will need to log in again before performing the next restore operation.

Step B5. Send the message `commitRestore`, which tells the system you are finished restoring transaction logs.

```
topaz 1> run
SystemRepository commitRestore
%
[Info]: Logging out session 2 at 8/20/11 16:21:47 PDT
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 GemStone full backup as soon as operational circumstances permit**.

To Restore Logs to a Point in Time

Ordinarily, the methods `Repository>>restoreFromLog:` 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** *fileName* **-I**. The maximum interval between checkpoints by default is five minutes. For example:

```
% copydbf tranlog2.dbf -I
Source file: tranlog2.dbf
  file type: tranlog  fileId: 2
  The file was created at: Tue 16 Aug 2011 10:50:53 PDT.
  The previous file last recordId is  -1.
  Scanning file to find last checkpoint...
Destination file:  /dev/null
  Checkpoint 1 started at: Tue 16 Aug 2011 15:33:50 PDT.
  oldest transaction references fileId -1 ( this file ).
  Checkpoint 2 started at: Tue 16 Aug 2011 15:38:50 PDT.
  oldest transaction references fileId -1 ( this file ).
  Checkpoint 3 started at: Tue 16 Aug 2011 15:43:40 PDT.
  oldest transaction references fileId -1 ( this file ).
  File size is 72704 bytes (142 records).
```

The following sequence restores the repository to the first checkpoint that would have included a commit on December 15 at 3:35:00 a.m.:

```
topaz 1> run
SystemRepository timeToRestoreTo:
 (DateTime fromString: '15/8/2011 15:35:00') .
SystemRepository restoreFromCurrentLogs
%
```

You can continue restoring past *aDateTime* by issuing another `restoreFromLog:` or `restoreFromCurrentLogs`. If you first issue another `timeToRestoreTo:`, restoration stops at the new *aDateTime*; otherwise, restoration continues to the end of the log.

Page Reclamation During Restore

The process of restoring transaction logs results in pages with shadow objects, which must either be reclaimed, or will cause the repository to grow. (For a detailed explanation of shadow objects and related concepts, see Chapter 10, "Managing Growth.") GcGems do not run during restore to handle this reclaim task in the background, so the Stone must reclaim these pages in the foreground, in addition to performing the restore itself.

The Stone configuration parameter STN_RECOVERY_PAGE_RECLAIM_LIMIT. controls the amount of reclaim work that is done. By default this is 2000, meaning that for every transaction log record replayed, the Stone will reclaim 2000 pages. Since each transaction log contains many, perhaps thousands, of transaction log records, this default ensures that all reclaim work is performed, and avoids any risk of excessive repository growth if there are a large number of shadow pages.

For fastest crash recovery or restore from backup, however, you may wish to do less reclaim, and accept more repository growth. In this case, set the STN_RECOVERY_PAGE_RECLAIM_LIMIT to a much smaller value — perhaps 5 or 10. Determining the best balance between performance and growth may require tuning. Also, some operations, such as indexing, markForCollection, and migrations, create a very large number of shadows, so be sure that the configuration you design allows for these situations.

For automatic recovery following a system crash, the configuration file setting for STN_RECOVERY_PAGE_RECLAIM_LIMIT will be used.

When you are restoring from backups, you may set configure the amount of reclaim programmatically using `System configurationAt: #StnRecoveryPageReclaimLimit put:` *anInteger*, prior to restoring transaction logs. You must be SystemUser to execute this statement.

Alternatively, you can restore transaction logs as quickly as possible, doing no reclaim, by using `restoreNoReclaimFromLog:` *fileOrDevice* instead of `restoreFromLog:` *fileOrDevice*. After the commitRestore, the garbage-collection Gem will reclaim the shadow pages in the background.

NOTE
Using `restoreNoReclaimFromLog:` *can cause the database to grow significantly, possibly running out of disk space.*

For warm standby systems, which run the `restoreFromLog:` operations continuously to keep a secondary repository updated from a primary's transaction logs, the balancing of restore performance and repository growth is more critical. For more on this, see "Managing Page Reclamation in Warm Standbys" on page 78.

Errors While Restoring Transaction Logs

Sometimes a transaction log is inadvertently truncated or corrupted. For instance, 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 the logs are being restored as a batch (such as `restoreFromCurrentLogs`), that method will try to recover by reading a replicate log.

If the missing transaction records are not found in a replicate log, or if you are restoring logs individually (`restoreFromLog:`), 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 `tranlog`*fileId*`.dbf`.

The method `restoreStatus` will return additional information indicating the next record expected within the current log file. For instance:

```
topaz 1> run
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to Tue 16 Aug 2011 14:52:06 PDT
next fileId = 37, record = 53
```

Retrieve a copy of the transaction log from an operating system backup, and complete restoring it by using `restoreFromLog:`*fileOrDevice*. Then continue the restore process by using either `restoreFromCurrentLogs` or another `restoreFromLog:`.

In the following example, `restoreFromCurrentLogs` encounters a truncated log (which happens to be in the first log). Invoking `restoreStatus` confirms that `tranlog1.dbf` is incomplete and shows that restoration needs to continue with record 33:

```
topaz 1> run
SystemRepository restoreFromBackup:'dbf/back.dat'
%
Restore from full backup completed with 39402 objects
restored and 0 corrupt objects not restored.
Ready for restore from transaction log(s).
topaz> login
...
successful login
topaz 1> run
SystemRepository restoreFromCurrentLogs
%
Gemstone: Error         Nonfatal
Restore from transaction log failed. Reason: 'Log with
fileId 1 is truncated or corrupt.
Continue restore with complete copy of this log.'
topaz>login
...
successful login
topaz 1> run
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to Tue 16 Aug 2011 14:52:06 PDT
next fileId = 1, record = 33
```

We recover by restoring a complete copy of that transaction log by name (here, "tranlog1.dbf.full").

```
topaz 1> run
SystemRepository restoreFromLog:'dbf/tranlog1.dbf.full'
%
Restore from transaction log succeeded.
topaz>login
...
successful login
topaz 1> run
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to Tue 16 Aug 2011 17:28:13 PDT
next fileId = 2
```

The restore status now shows that we are ready for the next transaction log, `tranlog2.dbf`. Because the remaining logs are online, we will return to our original procedure of restoring them as a group and then commit the restored repository:

```
topaz 1> run
SystemRepository restoreFromCurrentLogs
%
[Info]: Logging out session 2 at 08/20/11 16:18:21 PDT
Restore from transaction log(s) succeeded.
restoreFromBackup succeeded.
topaz>login
...
successful login
topaz 1> run
SystemRepository commitRestore
%
[Info]: Logging out session 2 at 8/20/11 16:13:46 PDT
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, you should read "Precautions When Restoring a Subset of Transaction Logs," next.

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, you can do that at any time after `restoreFromBackup:` by invoking `commitRestore`. Before doing that, however, you should be aware of the likely consequences and some precautions that may be appropriate:

1. Obviously, the omitted transactions will be lost. Unless the omitted log is missing, presumably that is your intent.
2. 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.

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 A9, on page 285) move all transaction logs (and log replicates, if used) to another directory.
2. After restoring from the backup, restore transaction logs individually by using `restoreFromLog:` (Step B2, on page 289) and providing the new path. Repeat this step for each log file you want to restore.
3. Invoke `commitRestore` to end the restore operation.

If you decide later to restore all logs by repeating the entire process, first remove any new log files and then move the previous transaction logs back to their original location. Follow the ordinary restore procedure.

9.4 How to Restore from an Operating System 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 Stone repository monitor was shut down. That is, the backup must have been started after you stopped GemStone and must have been completed before you restarted it.

If the file system itself has been corrupted, not just the extent files, see the section "Disk Failure or File System Corruption" on page 144.

NOTE
Backups created by an operating system dump of the whole system may or may not be usable. Before you consider restoring from system-wide backups, be sure to read "Why Operating System Backups May Not Be Usable" on page 275.

Step 1. If GemStone is still running, tell all users to log out. Use **stopstone** to stop the repository monitor.

Step 2. 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, all such files must be available.

WARNING
*Do **not** delete the transaction log files as of the crash — leave them online without moving them.*

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. 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 was in effect (STN_TRAN_FULL_LOGGING was set to False), the restoration process is complete. Restart GemStone by invoking **startstone** in the usual manner. You may now use your restored repository.

If full transaction logging was in effect (STN_TRAN_FULL_LOGGING was set to True), continue by restoring from transaction logs. Use **startstone -R** to restart GemStone. The -R option places the repository monitor in a state in which it is ready to restore from transaction logs, equivalent to that which follows restoration of a GemStone full backup.

To restore transaction logs, continue with the instructions under “B. To Restore Subsequent Transactions” on page 289

9.5 How to Recover After Repair of the File System

In case of a disk failure or a corrupt file system, the system manager must repair the file system 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, check the condition of the system repository with **pageaudit**. (See “How to Audit the Repository” on page 227 for instructions.)

- 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 the section “How to Restore a GemStone Repository” on page 281.)

To Recover When a File System Must Be Restored

If your system administrator intends to restore the file system from a backup device, before that happens it might be worthwhile to copy the repository to another node or to tape. 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:

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 227. You will need to specify the name and path of the copy using a temporary configuration file when doing **pageaudit** so that audit 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 `extent0.dbf` file that was restored from the file system backup. Check whether the backup was made while GemStone was running.

❒ If any changes were being made to the repository during the operating system backup, `extent0.dbf` may be an inconsistent file that cannot be made to work. In that case, you need to restore from a GemStone backup (see "How to Restore a GemStone Repository" on page 281). However, transaction logs from an operating system backup should be usable.

❒ If the operating system backup was done while GemStone was suspended or shut down, continue to the next step.

Step 2. Do a **pageaudit** to check the current (restored) `extent0.dbf` file. (See the section "To Perform a Page Audit" on page 227 for instructions.)

❒ 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 GemStone backups (see "How to Restore a GemStone Repository" on page 281).

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.*

9.6 Version compatibility

It is not always possible to restore backups made by a previous version of GemStone into a new version.

If you archive backups of your GemStone repository over multiple upgrades of your GemStone installation, you should also archive the GemStone executables for each version.

To determine the version of GemStone that a backup file was created with (in version 6.6 and later), use `copydbf -i backup.dbf`. See page 432 for details on copydbf.

9.7 Warm Standby Systems

GemStone's backup and restore mechanisms can be used to set up a secondary server, running almost in parallel with the primary server and ready to take over as quickly as possible in case of any failure of the primary system.

To do this, a backup of the primary server is restored into a separate location. This backup is left running in restore mode, and as the transaction logs are created on the primary server, they are immediately restored into the warm standby system. In case of failure of the primary system, the warm standby replays the final transaction log, executes a commitRestore, and is immediately ready to use.

For details on how to set up a warm standby system, see "How to Operate a Duplicate Server / Warm Standby" on page 75.

Chapter 10 Managing Growth

In the course of everyday operations, your GemStone 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/S 6.6 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. The following chapter, "Tuning Performance," describes various techniques for tuning both garbage collection and other aspects of your system.

This chapter discusses the following topics:

Basic Ideas
explains the main concepts underlying garbage collection.

Automatic Garbage Collection
describes the garbage collection mechanisms that occur automatically.

Invoking Garbage Collection
describes when, how, and why to invoke the garbage collection mechanisms that must be started by administrators.

10.1 Basic Ideas

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:

- identifiers for dead objects,
- the storage occupied by dead objects, and
- the storage occupied by shadow objects.

What Is Garbage?

Garbage consists of dead objects, shadow objects, and object identifiers (OOPs).

live object
GemStone considers an object *live* if it can be reached by traversing a path from AllUsers, the root object of the GemStone/S 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 object

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 object

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's 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 record

Views of the repository are based on *commit records,* structures written when a transaction is committed. Commit records detail every object viewed *(the read set)* and 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 126321) that refers to two other objects, its instance variables key (#Cost, oop 168165), and value (5.75, oop 126309).

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 .
%
[126321 sz:2 cls: 1745 SymbolAssociation] a SymbolAssociation
  key       [168165 sz:4 cls: 1733 Symbol] Cost
  value     [126309 sz:8 cls: 1521 Float]  5.7500000000000000E+00
topaz 1> commit
Successful commit
```

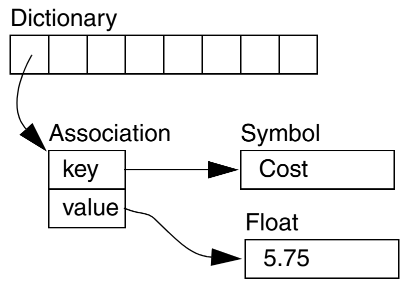


Figure 10.2 shows a second Topaz session that logs in at this point. Notice the Topaz prompt identifies the session by displaying a digit. Because Session 1 committed the SymbolAssociation to the repository, Session 2 can see the SymbolAssociation.

Figure 10.2 A Second Session Can See the Association

```
topaz 2> display oops
topaz 2> printit
Published associationAt: #Cost .
%
[126321 sz:2 cls: 1745 SymbolAssociation] a SymbolAssociation
  key       [168165 sz:4 cls: 1733 Symbol] Cost
  value     [126309 sz:8 cls: 1521 Float]  5.7500000000000000E+00
topaz 2>
```

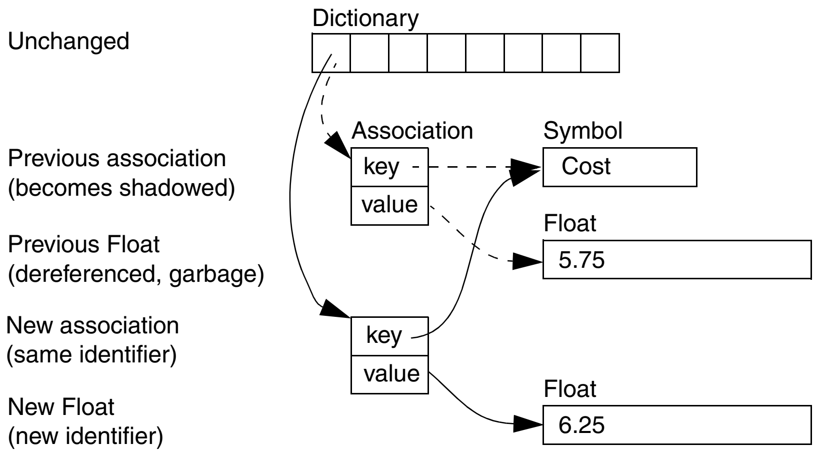
Now Session 1 changes the *value* instance variable, creating a new SymbolAssociation (Figure 10.3). Notice in the oops display that the new SymbolAssociation object has the same identifier (126321) as the previous Association.

- 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 126309) 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.

Figure 10.3 The Value Is Replaced, Changing the Association

```
topaz 1> printit
Cost := Cost + 0.50 .
Published associationAt: #Cost .
%
[126321 sz:2 cls: 1745 SymbolAssociation] a SymbolAssociation
  key      [168165 sz:4 cls: 1733 Symbol] Cost
  value    [126417 sz:8 cls: 1521 Float] 6.2500000000000000E+00

topaz 1> commit
Successful commit
```



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 .
%
[126321 sz:2 cls: 1745 SymbolAssociation] a SymbolAssociation
  key       [168165 sz:4 cls: 1733 Symbol] Cost
  value     [126309 sz:8 cls: 1521 Float]  5.7500000000000000E+00

topaz 2> abort

topaz 2>  printit
Published associationAt: #Cost .
%
[126321 sz:2 cls: 1745 SymbolAssociation] a SymbolAssociation
  key       [168165 sz:4 cls: 1733 Symbol] Cost
  value     [126417 sz:8 cls: 1521 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/S 6.6 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. Later 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.

   NOTE
   The Stone holds the set in memory until the next checkpoint; if the system should fail at this point, these steps would have to be redone.

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 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 *GcGem,* a process dedicated to 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. If any of those objects are indexed collections, however, additional work must be done. And if any are compiled methods, other Gems currently logged into the system will have to be signaled to reload their method caches. The GcGem now hunts through the set, looking for these special cases; when it finds them, it performs its job as necessary.

7. The Stone now promotes the objects from possibly dead to dead.

8. The GcGem 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.

9. OOPs are returned to their free pool.

NOTE
For information about tuning each step of garbage collection, see "Garbage Collection for Tuners" on page 376.

Different Ways to Collect Garbage

Garbage collection mechanisms vary according to:

- where garbage collection occurs—temporary (scratch) memory or permanent object space,
- how it occurs—automatically, or in response to an administrator's action, and
- what is collected.

Where

Each Gem session has its own private memory intended for scratch space, known as *local object memory.* This it uses for a variety of temporary objects, which can be garbage collected individually.

NOTE
For a detailed description of the structure of local object memory and its garbage collection process, see "Excessive In-Gem Garbage Collection" on page 372.

Permanent objects are organized in units of 8 KB called *pages.* Pages exist in the Gem's private page cache, the repository's 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 or dead object triggers reclamation of the page on which it 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. Another garbage collection mechanism is scheduled automatically, but is configurable.

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. With a bit of trial and error, you can find the right subset of these to run at intervals appropriate for your operational profile.

What

The main garbage collection mechanisms operate on whatever garbage they find in their universes. But if you have identified a specific kind of problem, you can invoke garbage collection processes that will operate only on shadow objects, or only on dead objects. With parallel resources, you can run both processes at the same time.

Sites with enough CPUs, I/O bandwidth, and multiple extents can run several such processes on specified extents, or on all extents, at the same time.

GemStone's Six Ways

GemStone 6.6 provides six mechanisms that together mark and reclaim garbage:

Both Marking and Reclaiming

Local 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 either a committed transaction or an overflow of temporary object space. These objects are collected individually, instead of in pages. This mechanism occurs automatically and cannot be configured.

NotConnectedSet collection — This mechanism examines the set of objects that the Gem session had to move to pages, even though they are not referenced by permanent objects. Each Gem performs a mark-sweep on this *notConnectedSet* when configurable thresholds are reached; it then releases the dead objects for the next garbage collection in permanent object memory. This mechanism also occurs automatically, but unlike the first, it can be configured. (Instructions are provided starting on page 372.)

Marking Only

Epoch garbage collection — The third mechanism examines objects shortly after they are committed to pages. Periodically, the 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. Various aspects can be configured to maximize its usefulness. (Basic instructions are provided on page 316, and more information is on page 366.)

Targeted marking — If you have reason to believe that specific objects are dead, you can invoke garbage collection specifically to mark them. These objects are added to the global queue GcCandidates, and the administrator initiates their collection by invoking `Repository>>markGcCandidates` in a GemStone Smalltalk session. This method sweeps the repository and marks those objects in GcCandidates that have no other references to them.

Repository-wide marking — The broadest mechanism must be invoked occasionally by a GemStone administrator to reclaim any dead objects that slip through the other collection mechanisms: `Repository>>markForCollection`. Because this method marks dead objects anywhere in the entire repository, it finds objects that elude detection with any other mechanism.

Reclaiming Only

Parallel reclamations in all specified extents — If you've traced a problem to an excess of shadow objects or an excess of dead objects, you can launch processes specialized to reclaim either shadow or dead objects. These specialized GcGems do not identify garbage; one reclaims pages with dead objects, and the other reclaims pages with shadow objects. They are not compatible with any other garbage collection mechanism, however.

Table 10.1 summarizes these different mechanisms:

Table 10.1 Garbage Collection Mechanisms Compared

Mechanism	What?	How?	Configurable?	Mark or Reclaim?	GcGem?
Local object memory collection	ephemeral objects in Gem's memory	automatic	no	both	not involved
NotConnectedSet collection	ephemeral objects in Gem's private page cache	automatic	yes	both	not involved
Epoch marking	dead and shadow object storage and OOPs in repository	automatic	yes	mark only	one of its duties
Targeted marking	dead and shadow object storage and OOPs among identified candidates	invoked	yes	mark only	finalizes voting, sweeps write set union
Parallel dead and shadow reclaim	either dead objects and OOPs, or shadow objects	invoked	yes	reclaim only	specialized GcGem
Repository-wide marking	dead and shadow object storage and OOPs in repository	invoked	yes	mark only	finalizes voting, sweeps write set union

10.2 Automatic Garbage Collection

Collecting garbage from local object memory and the notConnectedSet is an automatic process, though aspects of the notConnectedSet can be tuned for your operational profile.

Epoch garbage collection is also automatic; its default operation is controlled by configuration parameters in the GcGem's user account, GcUser—parameters you may wish to reconfigure once your applications have been running for a while and you have a good idea of what kind of garbage they produce, how much, and how often. Information on tuning epoch garbage collection is available on page 319 and also in "Inefficient Epoch Sweep" on page 366.

Collecting Local Object Memory

Local object memory exists in each session's private memory space. GemStone performs object garbage collection in local object memory to maximize its use and to minimize the number of dead objects that find their way to disk.

When a session creates a new object, GemStone allocates space for it in a work space in that Gem's local object memory. (Large objects, those occupying more than a single 8 KB page, are written directly in permanent object memory.) Objects that survive repeated generational scavenges in the work space eventually migrate to the Gem's *temporary object space* (also sometimes called *old space.)* The size of temporary object space is configurable; you may wish to tune it to the needs of your application.

If temporary object space fills, GemStone first performs a mark/sweep to remove any dead objects. If there is still not enough room, the next step is more drastic: objects in the first part of temporary object space are moved directly to permanent object memory, where they become part of the Gem's notConnectedSet. If the pages containing these objects are committed, future garbage collection must be handled by other, more costly techniques.

The GEM_TEMPOBJ_CACHE_SIZE configuration option determines the size of temporary object space, which by default is 585 KB. Sessions that create a large number of temporary objects may need to increase this to avoid cluttering permanent object memory with ephemeral objects.

For information on monitoring and tuning local object memory garbage collection, see "Excessive In-Gem Garbage Collection" on page 372.

Collecting the NotConnectedSet

After each generational scavenge of local object memory, each Gem examines the total number objects in its notConnectedSet—unreferenced objects that the Gem had to move to permanent object memory after temporary object space overflowed. When configurable limits are reached, the Gem performs a mark-sweep on the set. Any objects referenced from local object memory are kept in the notConnectedSet and protected from garbage collection. Remaining objects are garbage collected by the Gem, or (if the page has been committed) are marked for the next time pages are reclaimed.

Removing objects from a Gem's notConnectedSet provides two benefits:

- The objects can now be garbage collected while the session is still logged in.
- Voting overhead (page 310) is reduced because these objects have not become visible to other sessions.

For more information, see "Tuning the notConnectedSet" on page 373.

Epoch Garbage Collection

Epoch garbage collection operates on a finite set of recent transactions: the *epoch.* Using the write set the Stone maintains for each transaction, the GcGem examines every object created during the epoch. If it's unreferenced by the end of the epoch, it's marked as garbage and added to the list of possible dead. (Storage and identifiers are not reclaimed until the GcGem is reclaiming pages from other garbage as well.)

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 can't replace `markForCollection` because it will never detect objects created in one epoch and dereferenced in another.

Epoch garbage collection runs by default, but you can disable it, which you may wish to do for a period of time while you run other garbage collection mechanisms. You can also change how often it runs, or make it run at a time of your own choosing. After your installation has been operating for a while, and you've had the chance to collect operational statistics, you'll probably agree that epochs of the wrong length can be notably inefficient. An in-depth discussion of the performance trade-offs of short or long epochs starts on page 319. For more tuning information, see "Inefficient Epoch Sweep" on page 366.

Epoch garbage collection is one of several jobs that the GcGem performs. While it is running epoch garbage collection, the GcGem is occupied; its other jobs must wait. However, if too many pages need reclaiming, epoch garbage collection is deferred to free the GcGem for its page reclaim task.

The GcGem

To run epoch garbage collection, GemStone provides a specialized Gem, the GcGem, and a special predefined user account: GcUser.

The GcGem performs several tasks:

- It runs epoch garbage collection;
- finalizes the vote on possibly dead objects (Step 5, page 310);
- handles indexing structures, compiled methods, and other household tasks; and
- reclaims pages and OOPs.

While occupied with the write set union sweep and other household tasks (the middle two bullet items), the GcGem is unavailable for epoch garbage collection or page reclamation.

When STN_GC_SESSION_ENABLED is set to True (the default), GcUser logs in to the repository at GemStone startup through a Gem session process running as a child process of the Stone repository monitor. The GcUser session can be disabled in the GemStone configuration file, and certain Smalltalk methods stop the GcUser session temporarily by entering single-user mode.

Parameters to control epoch garbage collection and 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 `#GemIOLimit` to 100:

```
topaz> set user GcUser password thePassword
login
topaz 1> run
UserGlobals at: #GemIOLimit put: 100 .
System commitTransaction
%
```

Changes to GcUser's configuration parameters are not soon detected while the GcGem is executing a long-running operation, such as the write set union sweep. To change the I/O rate during execution of such operations, use the procedure described under "Changing the I/O Limit During a Long Operation" on page 90.

The following instance methods in `Repository` shut down the GcGem for their duration:

```
auditWithLimit:   repairWithLimit:
objectAudit
```

The method `Repository>>restoreFromBackup:` shuts down the GcGem when it begins. If it succeeds, the GcGem remains shut down until `restoreFromCurrentLogs` has successfully executed.

For information on tuning the GcGem, see "Overloaded GcGem" on page 370.

Collecting Statistics

Some cache statistics relevant to epoch garbage collection are:

- GcEpochState
- GcForceEpoch
- GcReclaimState
- ProgressCount

For more information on monitoring cache statistics, see "Monitoring Performance" on page 236.

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, and
- the epoch length *E*.

The only variables under your direct control is epoch length. You cannot specify it explicitly, together, these GcGem configuration parameters control the length of an epoch:

- `#epochGcEnabled`
  Enables (true) or disables (false) epoch garbage collection—by default, enabled.
- `#epochGcTimeLimit`
  The maximum frequency of epoch garbage collection—by default, 15 minutes.
- `#epochGcTransLimit`
  The number of transactions required to trigger epoch garbage collection—by default, 5000.
- `#epochGcByteLimit`
  The number of bytes of new or modified committed objects required to trigger epoch garbage collection—by default, 5 million.
- `#deferEpochReclaimThreshold`
  The number of pages needing to be reclaimed (another responsibility of the GcGem's) that will cause the GcGem to defer epoch garbage collection in favor of reclaiming pages—by default, 1000.

Epoch garbage collection occurs when:

```
(pages that need reclaiming < deferEpochReclaimThreshold) AND
(the time since last epoch > epochGcTimeLimit ) AND
( (transactions since last epoch > epochGcTransLimit) OR
(bytes committed since last epoch > epochGcByteLimit ) )
```

or when the method `System>>forceEpochGc` is executed

The following discussion assumes that the epoch is determined by the minimum time interval (`#epochGcTimeLimit`) because other thresholds are 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.

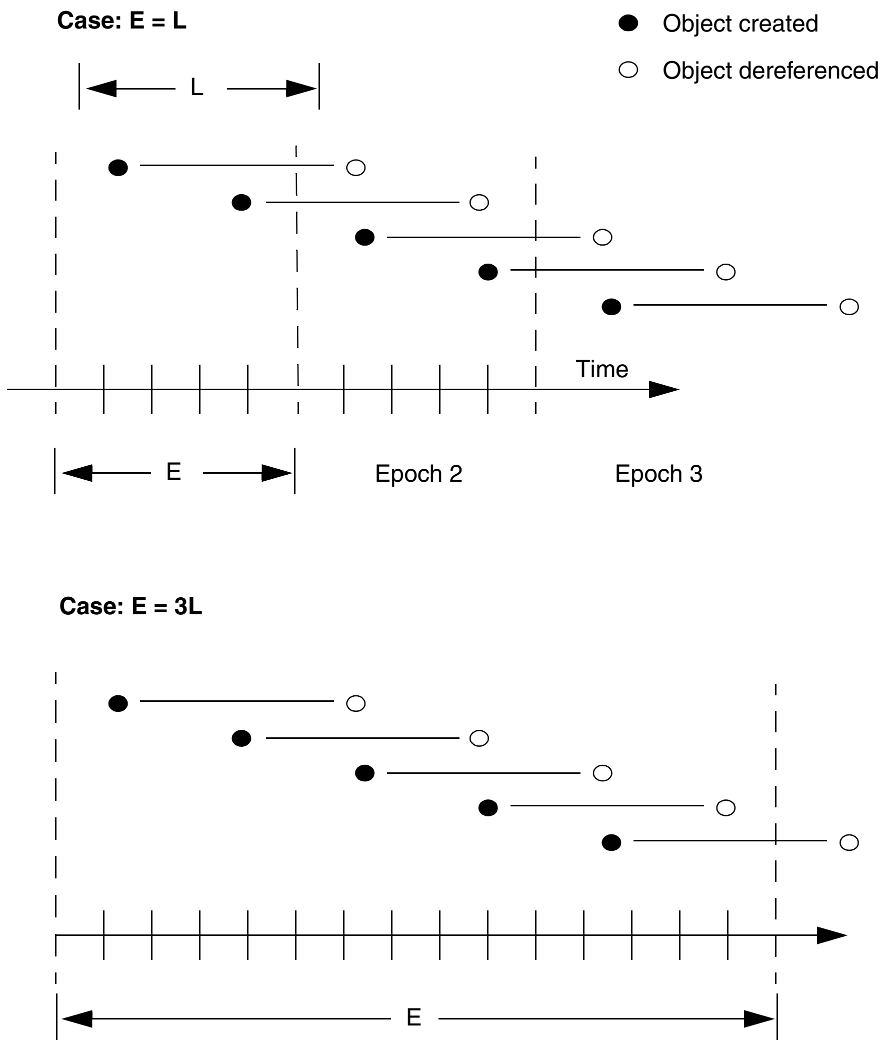
The results shown in Figure 10.5 can be expressed as:

Objects Missed by EpochGC	=	$R \times L$
Objects Recovered by EpochGC	=	$R(E - L)$

For example, assume R = 1000 objects per minute, L = 5 minutes, and E = 15 minutes. Then, for each epoch:

Objects Missed = 1000 x 5 = 5000
Objects Recovered = 1000 (15 – 5) = 10000

Figure 10.5 Effect of Collection Interval on Epoch Garbage Collection



Therefore:

❒ Set `#epochGcTimeLimit` E > lifetime L of short-lived objects

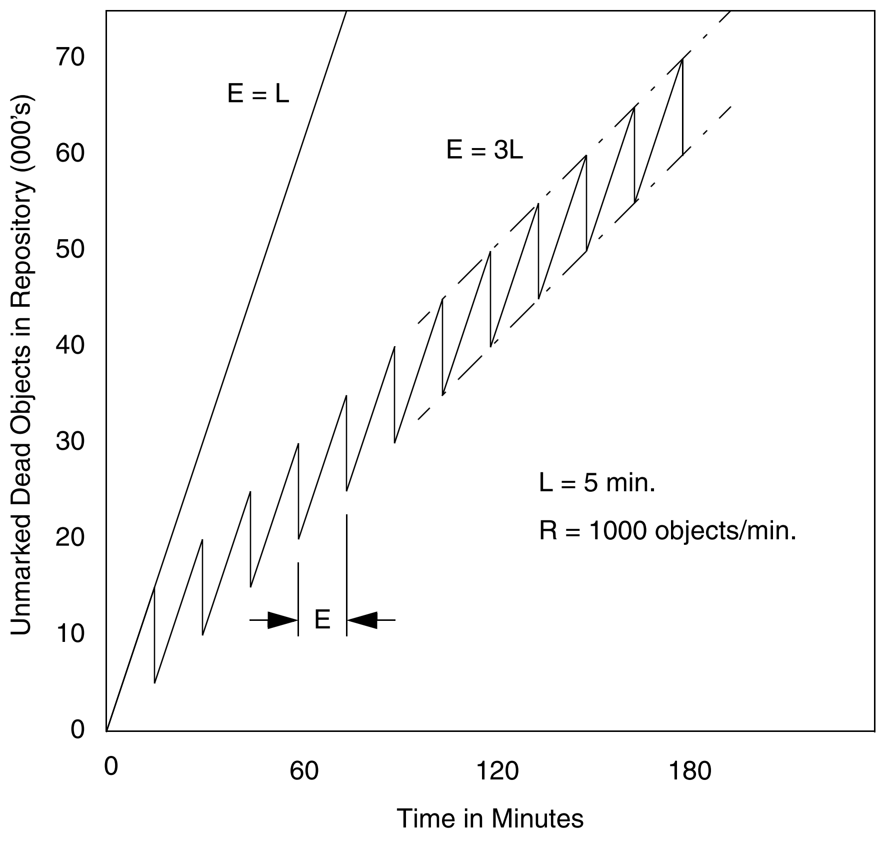
Figure 10.6 graphs the effect of the epoch. When $E = L$, epoch garbage collection is in effect disabled—all objects survive into the next epoch; the number of unmarked yet dead objects in the repository grows at the creation rate. These dead objects remain unidentified until you run `markForCollection`.

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:

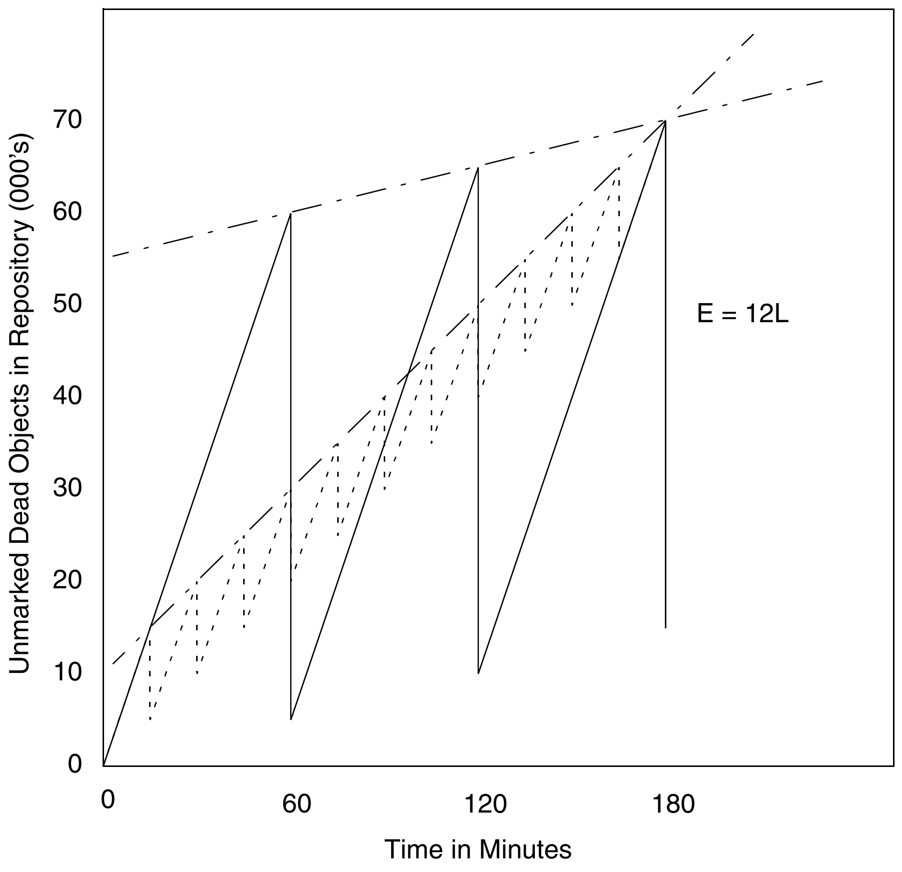
- moderate growth in the repository, and
- the need to run `markForCollection` often enough to mark dead objects that survive between epochs.

Figure 10.6 Repository Growth with Short Epoch



Suppose we extend the epoch to *E* = 12*L*. 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.

In some cases an epoch as long as several hours, or even a day, is appropriate.

Some application architectures commit objects that are used by a particular user's session, then dereference those objects when a user logs out. If most or all users log out at night, running an epoch GC once each night can be especially effective. To do this, you would set #epochGcTimeLimit to some value greater than 24 hours, and run a script each night that executes the method `System>>forceEpochGc`. For more information on the use of this method, see the comment on this method.

10.3 Invoking Garbage Collection

This section gives procedures and methods for controlling repository growth.

To prevent the repository from growing large enough to cause problems, on a regular basis, run:

```
Repository >> markForCollection
```

Instructions start on page 327.

If you can identify plausible targets for garbage collection yourself, you can supply hints to the system with:

```
Repository >> markGcCandidates
```

Instructions start on page 332.

To focus on reclaiming pages having either shadow or dead objects, or to run multiple reclamation processes at once, the following method starts either of two special kinds of GcGem:

`System Class >> startGC:` *gcSymbol*

One kind of GcGem reclaims storage from dead objects and object identifiers, as well as performing other tasks. The other kind reclaims storage from shadow objects. Instructions start on page 335.

If you have parallel resources and could benefit from multiple concurrent garbage collection processes on any or all of the extents, use the special kinds of GcGems started by either of these methods:

`System Class >> startGC:` *gcSymbol* `onExtent:` *extentId*
`System Class >> startGC:` *gcSymbol* `onExtents:` *extentIds*

For examples of how to invoke these methods, see “To Start or Stop GcGems Individually” on page 338.

To force the system to reclaim pages in general, you can use:

```
Repository >> reclaimAll
```

Instructions start on page 344.

If you find this plethora of choices a bit bewildering, see "Choosing When to Do What," below. This sketches a few typical scenarios and lists the incompatibilities between different mechanisms.

This section also provides a procedure for explicitly removing references to large objects, on page 348, and hints for collecting garbage collection candidates off-line, on page 350.

Choosing When to Do What

To begin with, for normal operations, we recommend that you run with the defaults appropriate to the size of your repository, as indicated by the sample configurations described starting on page 30. Depart from these defaults after analysis of your operations indicates a reason for doing so and a direction to go.

To aid in your analysis, GemStone gives you several tools, as well as the ability to collect a wide variety of statistics on many aspects of the system. These tools are described in Chapter 8, the statistics in "Cache Statistics" on page 239.

Additional tools you can use to view and analyze these statistics, as well as common performance bottlenecks, are described in Chapter 11.

Specialized GcGems are provided to deal with a large backlog of garbage awaiting reclaiming. Statistically, this situation evidences itself with high numbers for PagesNeedReclaimSize or DeadNotReclaimedSize.

Page backlogs can build up for several reasons: The GcGem may be spending too much time in epoch garbage collection, or handling voting, therefore falling behind on reclaiming pages. Or, perhaps you collect garbage daily, before the regularly scheduled full daily backup. If reclaiming does not complete before the backup starts, it's interrupted when the backup starts and not necessarily resumed when the backup completes. Unclaimed pages accumulate, and the next day's garbage marking process makes matters worse.

With four jobs, the generic GcGem has a lot to do and can sometimes be overloaded. The specialized GcGems are provided to help take some of the load off, by providing the ability to target system resources on specific kinds of page reclamation. But you cannot mix these GcGems indiscriminately. Their operations, and restrictions on those operations, are described starting on page 335.

MarkForCollection

The markForCollection operation is performed by executing the method `Repository>>markForCollection` or `Repository>>markForCollectionMt`. These mark ForCollection operations sweep the entire repository and mark as live all objects that can be reached through a transitive closure on the symbol lists in AllUsers, as described on page 304. 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 309. It does not reclaim the space or OOPs itself—the GcGem does.

Some objects may escape other garbage collection mechanisms, but the markForCollection is the broadest, identifying unreferenced objects in the entire repository.

However, markForCollection temporarily makes the repository larger. The repository grows in proportion to the total number of objects (live or dead) in the repository. In the worst case, this can add ((numberLive + numberDead) / 4) bytes. This bloat begins to decrease after the next checkpoint after markForCollection completes.

`Repository>>markForCollection` may be run on system that are in regular use, or systems that are offline or during off hours. For the fastest performance, you can run a multi-threaded version of markForCollection, `Repository>>markForCollectionMt`. This can complete the markForCollection operation faster, but should not be used in repositories that have any significant use, since there may be unacceptable commit record backlogs. Further information on multi-threaded markForCollection starts on page 331.

To Run markForCollection

Privileges required: GarbageCollection.

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> run
SystemRepository markForCollection
%
```

If you are performing `markForCollection` on a large production repository, consider the steps described under “Reducing Impact on Other Sessions” on page 329.

When `markForCollection` completes successfully, the Gem that started it displays a message such as the one below:

```
Successful completion of markForCollection.
    16482 live objects found.
    12 dead objects, occupying 2483 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.

NOTE
If you have uncommitted changes, this method does not abort, but returns an error. You must manually commit or abort your changes before you reattempt to run `markForCollection`.

Handling sigAborts

Because it runs outside of a transaction, `markForCollection` must respond to RT_ERR_SIGNAL_ABORT messages from the Stone. To avoid excessive interference with the marking process, consider temporarily raising the #StnSignalAbortCrBacklog internal parameter 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 67.

Conflicts with other sessions performing garbage collection

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.

```
The Garbage Collection process detected a concurrency 
conflict, reason:
#Garbage collection in progress by another session, try 
again later.
```

This concurrency conflict has several possible causes:

- An epoch garbage collection is in progress in a GcGem.
- A `markGcCandidates` or `markForCollection` is in progress in another session.

- A previous epoch, `markGcCandidates`, or `markForCollection` completed the mark phase, but voting on possibly dead objects has not completed.

  NOTE
  For voting to complete, either a GcGem or an EpochGem must be running. 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 either the generic GcGem (#GC) or the specialized EpochGem (#EPC) is running to complete processing of dead objects once the vote is completed.

  NOTE
  The GcUser parameter `deferEpochReclaimThreshold` *suppresses promoting possibly dead objects to dead (Step 7 on page 310) when too many pages need reclaiming, so that the GcGem can concentrate on reclaiming pages. Therefore,* `markForCollection` *will not complete until the GcGem has reclaimed enough pages to bring the value of* `deferEpochReclaimThreshold` *down below the threshold you have set.*

You can determine which session is performing a garbage collection operations using **`System class >> sessionIdHoldingGcLock`**.

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. You can do so in several ways:

- If the Gem session process performing `markForCollection` consumes too much of the available disk I/O resources, its I/O rate can be limited. Set the GEM_IO_LIMIT configuration option in a special configuration file read by that session, or change the `#GemIOLimit` internal parameter. As a starting point, determine your system's maximum disk I/O rate and the average rate under ordinary GemStone operation. Then set the Gem I/O limit for that session to (maximumRate – averageRate).

  For example, to set `#GemIOLimit` to 100, log in as the user whose Gem is running `markForCollection` and execute:

  ```
  topaz 1> run
  UserGlobals at: #GemIOLimit put: 100 .
  System commitTransaction
  %
  ```

- Have the Gem running `markForCollection` use a sufficiently large private page cache so that it does not interfere with use of the shared page cache by other sessions. Create a special configuration file for that Gem and increase the setting of GEM_PRIVATE_PAGE_CACHE_KB. As a starting point, determine the approximate number of objects in your repository. Object audits give this statistic, or you can estimate it by taking the size of your repository and allowing 100 bytes per object. Then use (numberOfObjects / 4000) as GEM_PRIVATE_PAGE_CACHE_KB. The maximum size permitted is 524288 KB.

- 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 3000 while `markForCollection` runs is a good place to start. However, if this large value causes out-of-memory errors, use a lower value.

  NOTE

  A sigAbort causes `markForCollection` *to redo its recent work, which in the worse 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.

- If excessive CPU usage seems to be the problem, you can reduce usage by using the UNIX **nice** command with the Gem process or linked application that is running `markForCollection`.

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.

To Run multi-threaded markForCollection

Privileges required: GarbageCollection.

Multi-threaded markForCollection, `Repository>>markForCollectionMt`, is similar to `markForCollection`, but has additional tuning parameters that can be specified as arguments.

`markForCollectionMt` can only be run from a gem that is local to the Stone (that is, the gem process is running the same machine as the stone), and cannot be run on Windows. You can determine if the session can run a `markForCollectionMt` by using the method `Repository>>supportsMtGc`.

`markForCollectionMt` is performed using two page buffers. By default, each page buffer size is 320, and each buffer uses two read threads. So by default, `markForCollectionMt` executes using a total page buffer space of 640, and 4 read threads, in addition to the main thread and a statistics thread.

For faster performance on more powerful machines, you can increase the number of pages per buffer and the number of threads per buffer. Note that since the arguments are per buffer, the total will be twice that; keep this in consideration to avoid running out of address space.

Like regular `markForCollection`, `markForCollectionMt` will respond to SigAbort requests from the stone; however, it may take significantly longer, particularly when using larger buffer sizes. markForCollectionMt should be run in repositories that are offline or under only light use (with only a limited commit rate), to avoid unmanageable commit record backlogs.

To execute multi-threaded markForCollection, log in to GemStone and send your repository the message `markForCollectionMt`, as in the following example:

```
topaz 1> run
      SystemRepository markForCollectionMt
%
```

This will execute the `markForCollectionMt` using four read threads (2 per buffer) and 640 total page buffer space (320 pages per buffer).

As regular markForCollection, this method will wait for up to two minutes for another session executing a garbage collection operations to complete. See the discussion on page 328. You may use the method `Repository>>markForCollectionMtWait:` *seconds* to specify how long to wait to being a multi-threaded markForCollection.

To execute the multi-threaded markForCollection using a total page buffer space of 2048 and 16 read threads, execute the following;

```
topaz 1> run
System Repository
      markForCollectionWaitMt: 120
      pageBufferSize: 1024
      threads: 8
%
```

Scheduling markForCollection

To invoke `markForCollection` using the **`cron`** facility, create a three-line script file similar to the Topaz example on page 327 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.

To Run markGcCandidates

Privileges required: GarbageCollection.

The method `Repository>> markGcCandidates` depends on the application to provide an array of candidate objects that it believes can be garbage-collected. The application does that by sending the message `Repository>>addGcCandidates:`*anArray*, which adds the array to the global queue GcCandidates. Because `markGcCandidates` depends on the application to provide a list of objects as hints to be swept for references, it is useful only if you have such hints.

By invoking `Repository>>markGcCandidates`, the administrator can have GemStone analyze the objects in GcCandidates and garbage collect those that are otherwise unreferenced. This method requires the GarbageCollection privilege.

Unfortunately, guessing wrong can be both time- and CPU-intensive. In the context of `markGcCandidates`, the definition of a dead object is more restrictive than it is in other contexts: `markGcCandidates` considers an object dead if it is referenced by no other object outside the candidates array, even a dead object. Therefore, if you omit a single one of a network of dead objects, none of the other dead objects in the network will be garbage-collected, because a single dead object outside the candidates array references them. Instead, they will all have to wait for repository-wide garbage collection.

The method performs an optimized linear sweep of the repository to identify any objects in GcCandidates that still have references. The remaining objects in GcCandidates become the list of possible dead objects. A transitive closure is performed only on the live objects found in GcCandidates. As a result, this method can identify possibly dead objects more efficiently than `markForCollection`, which performs a transitive closure on the entire repository.

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 309. Marking does not by itself reclaim space or identifiers.

The method `Repository>>markGcCandidates` marks any otherwise unreferenced objects in the global queue GcCandidates (garbage collection candidates). This method is more efficient than `markForCollection` in garbage collecting specific objects.

An application can add objects to the GcCandidates queue while archiving or dereferencing objects at the end of their life cycle. The application does that by sending the message `Repository>>addGcCandidates:`*anArray*. After the transaction has been committed, a user with the GarbageCollection privilege can then invoke this method to start the collection activity. The reference from GcCandidates is a *weak reference:* it does not keep `markGcCandidates` from marking the candidates

To check the queue, send the message `value` to the global object GcCandidatesCount. For example:

```
topaz 1> run
GcCandidatesCount value
%
500
```

To perform `markGcCandidates`, send that message to the repository:

```
topaz 1> run
SystemRepository markGcCandidates
%
```

This method aborts your current transaction and empties GcCandidates into a temporary, hidden array, then commits the transaction so that there are no committed references to the objects. The method then performs the garbage collection analysis while outside of a transaction.

NOTE
If you have uncommitted changes, this method does not abort, but returns an error. You must manually commit or abort your changes before you reattempt to run `markGcCandidates`.

Because the hints are in a temporary array at this point, ensure that the Gem process running `markGcCandidates` isn't terminated prematurely, or this work will have to be repeated.

When the method completes successfully, it returns an array containing the number of possibly dead objects found, the number of bytes they occupy, and any entries in GcCandidates that were not eligible for collection. For example:

```
an Array
  #1 500
  #2 4010000
  #3 first entry not eligible for collection
...)
```

If another garbage collection (epoch or `markForCollection`) is in progress at the time you try to do `markGcCandidates`, it may report a conflict error similar to that shown below. Try `markGcCandidates` again after a few minutes.

```
The Garbage Collection process detected a concurrency 
conflict, reason:
#Garbage collection in progress by another session, try 
again later.
```

NOTE
The GcUser parameter `deferEpochReclaimThreshold` *suppresses promoting possibly dead objects to dead (Step 7 on page 310) when too many pages need reclaiming, so that the GcGem can concentrate on reclaiming pages. Therefore,* `markGcCandidates` *will not complete until the GcGem has reclaimed enough pages to bring the value of* `deferEpochReclaimThreshold` *down below the threshold you have set.*

GcCandidates Removed from Indexed Collections

If your application adds an object to GcCandidates that was removed from an indexed collection, part of the indexing structure can retain a reference to that object, thus preventing its garbage collection. To avoid this, in the Globals symbol dictionary, set the global `#AddSystemObjectsToGcCandidates` equal to `true` (the default is false). Commit the transaction to make the value persistent and visible to others. You can now run `markGcCandidates` with the intended effect.

NOTE
To prevent GcCandidates from growing excessively when the global `#AddSystemObjectsToGcCandidates` *is* `true`*, you must run* `markGcCandidates` *more often than usual; indexing structures can be large.*

GcGems Specialized to Reclaim Pages

GemStone/S provides several specialized GcGems in addition to the original GcGem:

GcGem

The original GcGem, as described starting on page 317. It reclaims pages with shadow objects and dead objects repository-wide, and it finalizes voting and performs other household tasks.

Symbols: `#GcGem`, `#GC`

ReclaimGem

A GcGem dedicated to the task of reclaiming shadowed pages and dead objects repository-wide.

Symbols: `#ReclaimGem`, `#RCL`

EpochGem

A GcGem dedicated to the task of performing epoch garbage collection and finalizing voting on dead objects repository-wide.

Symbols: `#EpochGem`, `#EPC`

ParallelShadowReclaim

A GcGem dedicated to the task of reclaiming pages with shadow objects, on a specified extent. You can run one of these GcGems on every extent, if you have the hardware to make this an efficient choice.

Symbols: `#ParallelShadowReclaim`, `#PSR`

ParallelDeadReclaim

A GcGem dedicated to the task of reclaiming pages with dead objects, on a specified extent. You can run one of these GcGems on every extent, if you have the hardware to make this an efficient choice.

Symbols: `#ParallelDeadReclaim`, `#PDR`

Together, the ReclaimGem and EpochGem cover the full set of GcGem operations. However, this is not true for the parallel reclaim processes.

The ParallelDeadReclaim GcGem does not perform the GcGem's tasks of epoch garbage collection, finalizing the vote and special object processing. Therefore, `markForCollection` and `markGcCandidates` will be held up in the voting step of garbage collection. And you still need to run one or both of these methods periodically, because nothing else will identify objects created in one epoch and dereferenced in another.

The ParallelDeadReclaim GcGem has another limitation: while it's running, other users on the system can read the repository, but they cannot commit any changes to it. If they try, the commit fails with the transaction conflict `#CommitsDisabled`.

NOTE
Users cannot commit changes to the repository while the ParallelDeadReclaim GcGem is running. Use it only at times when a read-only repository can be tolerated.

Table 10.2 shows which kinds of GcGem can run at the same time:

Table 10.2 GcGem Compatibility

GcGem Kind	Compatible GcGems
Default GcGem	None
ReclaimGem	EpochGem
EpochGem	ReclaimGem
ParallelShadowReclaim	ParallelShadowReclaim on other extents
ParallelDeadReclaim	ParallelDeadReclaim on other extents

Because the ReclaimGem / EpochGem pair divides up the work accomplished by the original GcGem, running these two together can be more effective on a multiple processor machine.

Multiple ParallelShadowReclaim or multiple ParallelDeadReclaim GcGems can be run, one on each extent file in the repository. This can be useful if you have multiple extents and multiple processors.

Managing Garbage Collection Automatically

When the system configuration option STN_GC_SESSION_ENABLED is set to True, or when the dynamic configuration option `#GcSessionEnabled` is set to 1, the Stone automatically tries to keep GcGems running, according to the value set for the system configuration option STN_GC_SESSION_CONFIGURATION. Valid values are:

1 Keep a single GcGem running (the default).

This configuration is recommended for light garbage collection loads.

2 Keep a ReclaimGem / EpochGem pair running.

This configuration is recommended for moderate or heavy garbage collection loads. To benefit from this configuration, multiple processors are needed.

If a GcGem fails or is stopped by a message send, the Stone tries to restart it within two minutes:

- For STN_GC_SESSION_CONFIGURATION = 1:

  The GcGem is restarted if no other GcGems are running.

- For STN_GC_SESSION_CONFIGURATION = 2:
    - The ReclaimGem is restarted if no other GcGems are running except the EpochGem.
    - The EpochGem is restarted if no other GcGems are running except the ReclaimGem.

To prevent the stone from trying to start a GcGem that is not appropriate for a special configuration, set `#GcSessionEnabled` to 0. Remember to return `#GcSessionEnabled` to 1 when you're ready to resume automatic garbage collection.

To Start or Stop GcGems Individually

Under certain circumstances, it's desirable to control GcGems individually using methods on Class System. For instance, when a high number of pages need to be reclaimed, you may wish to shut down your ordinary GcGems during off hours and start up ParallelShadowReclaim GcGems on selected extents. For further information, see "Example GcGem Configurations" on page 341.

This section explains various methods for starting and stopping selected GcGems and for getting information about those that are running.

Methods that start or stop a GcGem return immediately, but the system action occurs in the background. Before taking action that requires the presence or absence of a GcGem, confirm its state by sending:

`System class>>gcSession:` *gcSymbol*`.`

Starting a Repository-Wide Special GcGem

Privileges required: GarbageCollection.

To start a special GcGem to reclaim pages from shadow and dead objects repository-wide, use:

`System startGC:` *gcSymbol*

gcSymbol can be any of the symbols `#GcGem`, `#ReclaimGem`, `#EpochGem` (`#GC`, `#RCL`, `#EPC`). These start the specified kind of GcGem, as described on page 335.

If *gcSymbol* is `#ParallelShadowReclaim` (`#PSR`) or `#ParallelDeadReclaim` (`#PDR`), the specified kind of GcGem starts on each extent.

Stopping a Repository-Wide Special GcGem

To stop the special GcGem, use:

`System stopGC:` *gcSymbol*

If *gcSymbol* is `#GcGem`, `#ReclaimGem`, or `#EpochGem` (`#GC`, `#RCL`, `#EPC`), stops the specified GcGem.

If *gcSymbol* is `#ParallelShadowReclaim` (`#PSR`) or `#ParallelDeadReclaim` (`#PDR`), stops GcGems of that type on all extents on which they are running.

Starting Extent-Specific Special GcGems

To start parallel special GcGems on specified extents, use:

`System startGC:` *gcSymbol* `onExtent:` *extentId*

gcSymbol can be any of the symbols `#GcGem`, `#ReclaimGem`, `#EpochGem`, `#ParallelShadowReclaim`, `#ParallelDeadReclaim` (`#GC`, `#RCL`, `#EPC`, `#PSR`, `#PDR`). These start the specified kind of GcGem, as described on page 335.

extentId can be any integer specifying a valid extent number.

If *gcSymbol* is `#GcGem`, `#ReclaimGem`, or `#EpochGem` (`#GC`, `#RCL`, `#EPC`), starts up the specified kind of GcGem, ignoring *extentId*.

If *gcSymbol* is `#ParallelShadowReclaim` (`#PSR`) or `#ParallelDeadReclaim` (`#PDR`), starts up the specified kind of GcGem on the specified *extentId*.

You can also use the method:

`System startGC:` *gcSymbol* `onExtents:` *extentCollection*

gcSymbol can be any of the symbols `#GcGem`, `#ReclaimGem`, `#EpochGem`, `#ParallelShadowReclaim`, `#ParallelDeadReclaim` (`#GC`, `#RCL`, `#EPC`, `#PSR`, `#PDR`). These start the specified kind of GcGem, as described on page 335.

extentCollection can be a Collection of SmallIntegers that are extentIds, or `#ALL`.

If *gcSymbol* is `#GcGem`, `#ReclaimGem`, or `#EpochGem` (`#GC`, `#RCL`, `#EPC`), starts up the specified kind of GcGem, ignoring *extentCollection*.

If *gcSymbol* is `#ParallelShadowReclaim` (`#PSR`) or `#ParallelDeadReclaim` (`#PDR`), starts up the specified kind of GcGem on the extents specified in *extentCollection*.

If *extentCollection* is `#ALL`, starts up the specified kind of GcGem on all extents.

Stopping Extent-Specific Special GcGems

To stop a special GcGem, use:

`System stopGC:` *gcSymbol* `onExtent:` *extentId*

gcSymbol can be any of the symbols `#GcGem`, `#ReclaimGem`, `#EpochGem`, `#ParallelShadowReclaim`, `#ParallelDeadReclaim` (`#GC`, `#RCL`, `#EPC`, `#PSR`, `#PDR`). These stop the specified kind of GcGem, as described on page 335.

extentId can be any integer specifying a valid extent number.

If *gcSymbol* is `#GcGem`, `#ReclaimGem`, or `#EpochGem` (`#GC`, `#RCL`, `#EPC`), stops the specified GcGem, ignoring *extentId*.

If *gcSymbol* is `#ParallelShadowReclaim` (`#PSR`) or `#ParallelDeadReclaim` (`#PDR`), stops GcGems of the specified kind on the specified *extentId*. If you specify an extent on which the specified kind of GcGem is not running, the invalid extent number is ignored.

You can also specify extents by using a Collection:

`System stopGC:` *gcSymbol* `onExtents:` *extentCollection*

gcSymbol can be any of the symbols `#GcGem`, `#ReclaimGem`, `#EpochGem`, `#ParallelShadowReclaim`, `#ParallelDeadReclaim` (`#GC`, `#RCL`, `#EPC`, `#PSR`, `#PDR`). These start the specified kind of GcGem, as described on page 335.

extentCollection can be a Collection of SmallIntegers that are extentIds, or `#ALL`.

If *gcSymbol* is `#GcGem`, `#ReclaimGem`, or `#EpochGem` (`#GC`, `#RCL`, `#EPC`), stops the specified GcGem, ignoring *extentCollection*.

If *gcSymbol* is `#ParallelShadowReclaim` (`#PSR`) or `#ParallelDeadReclaim` (`#PDR`), stops all GcGems of the specified kind on the specified *extentCollection*. If *extentCollection* is `#ALL`, stops GcGems of the specified type on all extents. If you specify an extent on which the specified kind of GcGem is not running, the invalid extent number is ignored.

Getting GcGem Session Information

The following methods return the session IDs of the specified GcGems, or 0 if no GcGem of the specified type is currently running.

`System gcSession:` *gcSymbol*

If *gcSymbol* is `#GcGem`, `#ReclaimGem`, or `#EpochGem` (or `#GC`, `#RCL`, `#EPC`, respectively), returns the session ID of the specified kind of GcGem, or 0 if none.

If *gcSymbol* is `#ParallelShadowReclaim` (`#PSR`) or `#ParallelDeadReclaim` (`#PDR`), returns an array of session IDs corresponding to each extent.

If *gcSymbol* is `#ALL`, returns an array of session IDs in the following order:

1. GcGem
2. ReclaimGem
3. EpochGem
4. An array of session IDs, one for each extent, starting with extent 0 and ending with the highest extent number on which a ParallelShadowReclaim GcGem is running.
5. An array of session IDs, one for each extent, starting with extent 0 and ending with the highest extent number on which a ParallelShadowReclaim GcGem is running.

To obtain the session information about a specific extent, use:

`System gcSession:` *gcSymbol* `onExtent:` *extentId*

If *gcSymbol* is `#GcGem`, `#ReclaimGem`, or `#EpochGem` (or `#GC`, `#RCL`, `#EPC`, respectively), returns the session ID of the specified kind of GcGem (or 0 if none). The *extentId* is ignored.

If *gcSymbol* is `#ParallelShadowReclaim` (`#PSR`) or `#ParallelDeadReclaim` (`#PDR`), returns the session ID for the GcGem on the specified *extentId*.

You can also use:

`System gcSession:` *gcSymbol* `onExtents:` *extentArray*

If *gcSymbol* is `#GcGem`, `#ReclaimGem`, or `#EpochGem` (`#GC`, `#RCL`, `#EPC`), returns the session ID of the specified kind of GcGem (or 0 if none). The extent IDs are ignored.

If *gcSymbol* is `#ParallelShadowReclaim` (`#PSR`) or `#ParallelDeadReclaim` (`#PDR`), returns an array of session IDs for the GcGems corresponding to the extents in *extentArray*.

If *extentArray* is `#ALL`, returns an array of session IDs for all extents.

Example GcGem Configurations

This section first presents example configurations addressing the need to reclaim either a high number pages or a high number of objects. Finally, it presents guidance about other configurations.

Light Load

Run with a single normal GcGem (STN_GC_SESSION_CONFIGURATION = 1).

Moderate Load

When they become available, run with a ReclaimGem / EpochGem pair (STN_GC_SESSION_CONFIGURATION = 2).

To benefit from this configuration, you need multiple processors.

High Number of Pages to be Reclaimed

During normal hours, run with the ReclaimGem / EpochGem pair (STN_GC_SESSION_CONFIGURATION = 2). During off-hours, shut down those GcGems and start up ParallelShadowReclaim GcGems on the extents. For example:

```
System stopGC: #ALL.
System sleep: 5.  "To give current gems time to logout"
System startGC: #PSR onExtent: 0.
System startGC: #PSR onExtent: 1.
....
System startGC: #PSR onExtent: max_extent_id.
```

To benefit from this configuration, you need multiple processors and multiple extents.

NOTE
Remember to start the parallel shadow reclaim GcGems within two minutes of shutting down the current GcGems. Otherwise, the Stone will restart the ordinary GcGems.

Running with parallel shadow reclaim GcGems on all extents may put too high a load on the system; it might be preferable to run them on just a small number of extents at a time. Experiment to determine what works best on your system.

When the number of pages needing reclamation has returned to normal, return to your ordinary GcGem configuration by simply shutting down the parallel shadow reclaim GcGems. The Stone will restart the ordinary GcGems within two minutes.

High Number of Objects to be Reclaimed

During normal hours, run with the ReclaimGem / EpochGem pair (STN_GC_SESSION_CONFIGURATION = 2). During off hours, shut down those GcGems and start up parallel dead reclaim GcGems on the extents. For example:

```
System stopGC: #ALL.
System sleep: 5.  "To give current gems time to logout"
System startGC: #PDR onExtent: 0.
System startGC: #PDR onExtent: 1.
....
System startGC: #PDR onExtent: max_extent_id.
```

To benefit from this configuration, you need multiple processors and multiple extents.

NOTE
Remember to start the parallel dead reclaim GcGems within two minutes of shutting down the current GcGems. Otherwise, the Stone will restart the ordinary GcGems.

Also, commits are not allowed while any parallel dead reclaim GcGems are running. Other sessions can operate on the repository, but any commits attempted will fail.

Running with parallel dead reclaim GcGems on all extents may put too high a load on the system; it might be preferable to run them on just a small number of extents at a time. Experiment to determine what works best on your system.

Parallel dead reclaim GcGems automatically shut down when they have completed reclaiming all dead objects on their designated extent. The Stone then restarts ordinary GcGems within two minutes of the last parallel dead reclaim GcGem shutdown.

Other Configurations

If you wish to experiment with other configurations, disable automatic GcGem startup by setting either STN_GC_SESSION_ENABLED to False or `#GcSessionEnabled` to 0. Otherwise, the Stone will attempt to restart GcGem processes that could conflict with your goals.

NOTE
To prevent shadowed pages or dead objects from accumulating, monitor your system closely.

General Page Reclaim

When the STN_GC_SESSION_ENABLED configuration option is set to True, the GcUser session automatically starts to reclaim pages while user sessions are running. The reclaim task 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 (or yours and the GcGem's), you can invoke page reclamation directly. The procedure to do that is described next.
- If other users are logged in, you can determine which sessions are viewing the oldest commit record, thereby impeding reclamation. See "To Identify Sessions Holding Up Page Reclamation" on page 346.

NOTE
If you need to force page reclamation, we encourage you to use the specialized GcGems as described starting on page 335 instead of the generalized mechanism described below. The specialized GcGems are less error-prone and easier to use than the method `Repository>>reclaimAll` *described below.*

To Invoke Reclamation

Privileges required: GarbageCollection.

Reclaiming objects previously marked as dead can be done explicitly by invoking `Repository>>reclaimAll`.

NOTE
To invoke this method, you must be the only user logged in, besides the GcGem's session. If a GcGem is running, it must be either the generic

GcGem (#GC) or the ReclaimGem / EpochGem pair (#RCL and #EPC). A backup must not be in process.

Step 1. Make sure you are the only user logged in (other than the GcGem). See "How to Enter Single-User Mode" on page 188.

Step 2. Before running `reclaimAll`, you should abort your current transaction:

```
topaz 1> run
SystemRepository abortTransaction
%
```

Step 3. Send the message `reclaimAll` to the repository:

```
topaz 1> run
SystemRepository reclaimAll
%
true
```

When the method returns true, reclaiming will be complete. However, the space may not appear as free pages until after the next checkpoint.

Be sure to check the return value. False indicates that reclamation did not succeed, most likely because another user was already logged in.

This method suspends all logins until it completes. It behaves differently, depending on whether the GcGem is present or not:

- If the GcGem is present, the GcGem responds by reclaiming batches of pages at a time, the size of the batch being determined by the parameters `reclaimMinPages` and `reclaimMaxPages`. The GcGem commits its transaction after each batch. Page reclamation proceeds more slowly, but it can be interrupted if necessary.
- If the GcGem is not present, the Stone reclaims all the pages at once. Page reclamation occurs more quickly than in the above case, but cannot be interrupted. If the repository has many pages to reclaim, the process can nevertheless take quite a while, during which time the system is completely unresponsive. Interrupting or halting the Stone during this process can corrupt your repository.

CAUTION
While the Stone is reclaiming pages, it cannot be interrupted. If the repository has many pages to reclaim, the system will be completely unresponsive – possibly for a long time. Interrupting or halting the Stone during this process can corrupt your repository. For this reason,

we recommend that you use the specialized GcGems to reclaim pages instead. Instructions for their use start on page 335.

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.

Sometimes, therefore, it's 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 1
#2 2
topaz 1> run
System currentSessionNames
%
session number: 1    UserId: GcUser
session number: 2    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

Configuration parameters to control the reclaim task are stored as the following values in GcUser's UserGlobals:

`#reclaimSleepTime`	the maximum amount of time in seconds that the process will sleep when no work is scheduled; must be ≥ 3; the default is 10 seconds

`#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.
`#reclaimMinPages`	The minimum number of pages to process in a single reclaim operation (reclaiming does not start until this threshold is reached); must be ≥ 10; the default is 40 pages.
`#reclaimMaxPages`	The maximum number of pages to process in a single reclaim; must be ≥ (`reclaimMinPages` + 5); the default is 200 pages.
`#reclaimDeadEnabled`	A Boolean indicating whether or not to reclaim dead objects; the default is true.
`#deferEpochReclaimThreshold`	Causes the GcGem to defer epoch garbage collection while the reclamation backlog exceeds its value. Epoch garbage collection task competes with the reclaim task for GcGem resources and potentially adds pages to the backlog. The default is 1000 pages.

For an example of how to change these parameters, see page 317.

The method `System class>>cacheStatistics:` for the Stone's cache slot reports several statistics about the reclamation cycle:

DeadNotReclaimedSize	The number of objects known to be dead but not yet reclaimed.
PagesNeedReclaimSize	The amount of work waiting for the Reclaim task.
PossibleDeadSize	The number of objects marked as dereferenced but not yet declared to be dead.
ReclaimCount	The number of times the page scavenge (reclamation) process has been run.
ReclaimedPagesCount	The number of scavenged pages.

For more information about these statistics, see "Cache Statistics" on page 239.

In cases where the backlog PagesNeedReclaimSize is quite large, it may be desirable to increase `#reclaimMaxPages` so that more objects are reclaimed in a single run, if that does not adversely affect users. It may also be desirable to defer epoch garbage collection by reducing the `#deferEpochReclaimThreshold`. These changes allow more of the GcGem's time to be devoted to page reclamation.

To Remove References to Large Objects

If you know 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> run
AllUsers findObjectsLargerThan: aSize limit: aSmallInt
%
```

NOTE
This method aborts your transaction, as do most methods that scan the repository. If you have uncommitted changes, this method does not abort, but returns an error.

This method 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> run
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> run
anObject findReferencesWithLimit: aSmallInt
%
```

NOTE
This method aborts your transaction, as do most methods that scan the repository. If you have uncommitted changes, this method does not abort, but returns an error.

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> run
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.

To Identify Candidates Off-line

On a very large repository—for example, 100 GB—running `markForCollection` might take many days. For such cases, an alternate mechanism exists: you can find garbage collection candidates in an off-line copy of the repository, and use the result to run `markGcCandidates` on the operational repository, a much quicker operation.

The following procedure outlines the process in general terms. For specifics, contact GemStone/S Technical Support. You may also find it helpful to examine the `offlinegc` example in the `$GEMSTONE/examples` directory.

NOTE
If Epoch garbage collection is running, disable it now. Do not run any other garbage collection while working with an off-line repository.

Step 1. Make an off-line copy of the repository and start it.

Step 2. On the copy, log in as SystemUser (or any user with garbage collection privileges).

Step 3. Execute `SystemRepository >> findDisconnectedObjects` or `SystemRepository >> findDisconnectedObjsMtWithPageBufferSize:threads:`.

findDisconnectedObjects uses the same algorithm to find dead objects as markForCollection does; however, it returns the results as an array of objects.

Since this is an offline repository, we recommend using `findDisconnectedObjsMtWithPageBufferSize:threads:`, with as large values of page buffer size and number of threads as appropriate for your system. See "To Run multi-threaded markForCollection" on page 331 for a discussion of the arguments to multi-threaded garbage collection.

Step 4. Determine the object identifiers of the resulting objects and write them to a file.

Step 5. Transfer the file to the host machine on which your operational repository resides.

Step 6. At a time when system activity is relatively low, read the file, reconstitute each object from its identifier, and place the resulting array of objects into the GcCandidates set using `SystemRepository >> addGcCandidates:`, supplying as an argument the collection of reconstituted objects.

Step 7. Finally, execute:

```
topaz 1> run
SystemRepository markGcCandidates
%
```

This scans each object again to ensure that other garbage collection mechanisms running in the operational repository (such as epoch garbage collection) have not, in the interim, recycled one of OOPs and reassigned it to a new, live object. This scan also identifies the dead objects to the operational repository, which can now dispose of them using any appropriate garbage collection mechanism.

Chapter 11 Tuning Performance

This chapter tells you:

- how to determine if your system suffers from any of several common performance bottlenecks, and, if so, how to remedy the situation, and
- the salient points of garbage collection, if you have traced a performance problem to that part of the system.

Diagnosis of these or other problems requires understanding how to use statmonitor and VSD, the GemStone utility programs for measuring system performance, to monitor cache statistics. Cache statistics are described in detail in "Cache Statistics" on page 239. Appendix G, "statmonitor and VSD Reference," describes how to use VSD and statmonitor.

11.1 Common Performance Bottlenecks

The following most commonly experienced performance bottlenecks each have an associated template to help you determine if this is a problem for your application.

- The shared page cache is too small.
- The commit record backlog is too large.

- The page server is swamped.
- The sweep phase of epoch garbage collection is inefficient.
- The garbage collection Gem is overloaded.
- In-Gem garbage collection is excessive.

Shared Page Cache Too Small

Statistic	Default Filter	Process	Explanation
FreeFrameCount		shared page cache monitor	See page 248.
FramesFromFindFree	per second	all	See page 247.
FramesFromFreeList	per second	all	See page 247.

Background

The shared page cache is a representation of part of the repository in RAM, allowing the Stone and Gem processes to read objects from memory rather than disk, similar to the way many applications use virtual memory. Because memory access is significantly faster than disk I/O, applications can run faster if they find most of the objects they require already in the shared page cache.

Both on disk and in the cache, objects reside on eight-kilobyte pages. The shared page cache has a fixed number of 8192 byte (8 KB) *frames* (the precise number of frames depending on the cache size); each frame can hold one page.

When a Gem seeks an object, it first looks in the shared page cache; only if the object is not already there does it search on disk. When it finds the object, it must obtain a free frame before it can read the page into the shared page cache.

The shared page cache monitor maintains a list of cache frames not currently in use: the number of frames in this list is reported by the statistic **FreeFrameCount**. When the Stone or a Gem uses a frame to hold a page, the frame is taken off the free frame list. Frames are added to the free list at checkpoints, or when a Gem logs out. However, when a page server writes a page to disk (perhaps in response to a committed transaction), it does not always add the frame that contained that page to the free frame list right away. Instead, it waits until the number of free frames in the list is less than 12.5% percentage of the frames in the cache. Thus, the free frame list is not always up-to-date.

A Gem in need of a free frame can get one in either of two ways:

- It can take a frame from the free frame list. If it finds one, statmonitor increments the statistic **FramesFromFreeList**.
- If it doesn't find one, the Gem scans the cache for a free frame that hasn't yet been added to the free list. If it finds one, statmonitor increments the statistic **FramesFromFindFree**.

Which method the Gem uses to get a free frame is determined by the Gem configuration parameter GEM_FREE_FRAME_LIMIT, set in the Gem's configuration file or at runtime. The algorithm is:

```
if FreeFrameCount > GEM_FREE_FRAME_LIMIT
      then take frame from free list
else scan cache for frame
```

By default, GEM_FREE_FRAME_LIMIT specifies that each Gem is entitled to take from the free list until the free list falls below 10% of the frames in the shared page cache. The garbage collection Gem (GcGem) is a special case: it takes frames from the free list until the number of free frames falls below 5% of the frames in the shared page cache.

For example, a 100 MB shared page cache contains 12,800 frames (100 MB / 8 KB). By default, then, each Gem must scan the cache instead of taking from the free list, when the free list holds fewer than 1280 frames (10% of 12,800 = 1280). The GcGem won't have to scan the cache until the free frame list falls below 640 (5% of 1280).

If no free frame can be found with either method, the Gem looks for a dirty page and writes it to disk itself, thus freeing its frame. This is the only occasion for a Gem to write a page to disk, and it is expensive.

The Stone always has a free frame limit of 0, meaning the Stone can take frames from the free list until the last one is gone. The Stone needs free frames to process a commit; if this slows down, all users suffer slower performance.

Evaluation

Getting a frame from the free list is significantly faster than having to scan the entire cache for a free frame. Therefore, the ratio of `FramesFromFreeList` to `FramesFromFindFree` can reveal a significant opportunity for increasing performance for your application.

Another sign that the shared page cache is too small is that FreeFrameCount is almost always at or near the free frame limit.

Finally, if you see that Gems are doing `pageWrites`, your shared page cache is almost certainly too small, or the Gem's free frame limit is too high.

For the Stone, the value of `FramesFromFindFree` should always be zero. If it is not, the shared page cache is too small, or not being used appropriately.

Additional helpful statistics:

- **LocalPageCacheMisses** and **LocalPageCacheHits** are another way to determine whether a Gem needs free frames. LocalPageCacheMisses is incremented when a Gem tries to access an object that it has not yet used, and finds it is not in its private page cache. Because the object may be in the shared page cache (if some other process has already accessed it), this statistic can be somewhat misleading, but, in general, if the Gem experiences more misses than hits, it is trying to access objects frequently, and has frequent need to find frames.
- **PageReads** reports the number of times a Gem has to read a page from disk. This occurs every time it needs an object not already in its private cache, or in the shared page cache.
- **GlobalDirtyPageCount** is the number of dirty pages that the page server cannot yet write to the repository.
- **PageWrites** reports the number of times a Gem has to write a page to disk in order to free a frame, because it was unable to find a free frame either from the free list or by searching the shared page cache.

Solutions

If a Gem more frequently scans the shared page cache for free frames instead of finding them on the free list, consider starting one or more free list page servers. (A description and instructions are available in "Adding Page Servers" on page 71.) Add one at a time until FramesFromFindFree goes to zero, or near zero. Increasing the size of the shared page cache may also be helpful.

Worse, if a Gem is writing pages, consider increasing the size of the shared page cache. For details on setting the page cache size, see "To Set the Page Cache Options and the Number of Sessions" on page 40.

Another possible remedy could be redesigning your application so that it needs fewer pages. Clustering objects often used together on a single page, or choosing different data structures, can under some circumstances significantly reduce the number of pages the Gem requires.

NOTE
Gems can be configured individually. To do so, see "Naming Executable Configuration Files" on page 398.

Commit Record Backlog Too Large

Statistic	Default Filter	Process	Explanation
TotalCommits	per second	Stone	See page 270.
CommitRecordCount	per second	Stone	See page 245.
CommitQueueSize	per second	Stone	See page 244.

Background

A *commit record* is the structure the Stone uses to provide a Gem with its view of the repository. Every time a Gem commits a transaction, it creates a commit record—a list of the objects read and written by the Gem that created it, and other associated information. The Stone maintains a collection of commit records ordered from oldest to newest; when a Gem commits or aborts a transaction and gets an updated view of the repository, its view is the state of the persistent objects as represented by the latest commit record.

At any given time, each Gem is connected to exactly one commit record, but each commit record can be connected to more than one Gem. That is, if Gem A commits a transaction and Gem B aborts immediately thereafter, they'll both share the same view of the repository: the view represented by the commit record created by Gem A.

The *commit record backlog* is the collection of all commit records connected to all active Gems—that is, all active views of the system. Commit records are kept in the shared page cache, where they contend for frames along with other resources. A commit record cannot be removed from the shared page cache until:

- no Gem session is looking at the view it represents, and
- it is the oldest commit record.

Therefore, if a single Gem stays in a transaction for a long time without committing or aborting (thereby freeing its commit record), newer commit records cannot be removed, even if no Gem references them. If this situation remains unchanged for a long time, a significant commit record backlog can build up.

SigAbort signals

The Stone's only means of correcting this situation is to signal the Gem to abort its transaction, thereby releasing the commit record, permitting it (and perhaps other unreferenced commit records) to be removed from the shared page cache.

A Gem will receive this signal, called a SigAbort, if:

- it is outside a transaction, and
- it refers to the oldest commit record, and
- the commit record backlog is greater than the specified maximum.

If a Gem is in transactionless mode:

> It is never in a transaction and never needs to commit changes. In this case, GemStone/S handles SigAbort signals for you transparently.

If a Gem is in automatic transaction mode:

> It is never outside a transaction and therefore will never receive a SigAbort. Automatic transaction mode can therefore cause significant performance problems if Gems do not commit or abort frequently.

If a Gem is in manual transaction mode:

> You'll need to write a handler to catch SigAbort signals and respond appropriately. (The class GsSession has a generic SigAbort handler method, which you can customize by writing your own block.)

SigLostOTRoot signals

If a Gem receives a SigAbort and does not respond (by aborting and releasing the commit record) in the length of time specified by the STN_GEM_ABORT_TIMEOUT configuration parameter (by default, one minute), then the Stone does the following:

1. Sends a SigLostOTRoot signal to the Gem.
2. Waits the number of seconds specified in the configuration parameter STN_GEM_LOSTOT_TIMEOUT for the Gem to acknowledge receiving the signal and abort.

If the Gem does not acknowledge the SigLostOTRoot signal, and the value of STN_GEM_LOSTOT_TIMEOUT is zero or greater, the Stone (more precisely, the Page Manager) poisons the session's shared page cache slot (and the slot of the gem's page server, if any) and sets the stopSessionRequested flag. For a remote Gem, the

Page Manager directs the appropriate remote cache page server to poison the Gem's slot.

After the Page Manager finishes, the session is logged off. Thereafter, any attempt by the session or its page server to access the shared cache will raise a fatal error.

In three cases, the Page Manager is unable to set the stopSessionRequested flag:

- The session is doing a reclaimAll.
- The session is nameless.
- STN_GEM_LOSTOT_TIMEOUT = -1.

If the Stone cannot stop the session, it revokes the Gem's access to the old commit record, which can then be removed. At the same time, the Stone immediately retracts the session's commit record and poisons its cache slot (the session will get a fatal error if it references the shared cache).

Logging out a non-responsive Gem

When a session is first processed for logout, the logout can only be immediately completed if the Stone received a disconnect on the Gem's OOB socket. (For remote Gems, the Gem's page server process must also have terminated.) If these conditions are met, the logout is processed as normal. If not, then the Gem's commit record view is dereferenced and the session is placed on a logout wait queue. The Stone checks sessions in the wait queue every two seconds. A 60-second timer for the process is also started.

When the Stone checks the sessions in the wait queue and the 60-second timer has not yet expired, Stone will complete the logout of the session under the following conditions.

- For local Gems, the Stone has received a disconnect on the Gem's OOB socket, *or* the gem process ID no longer exists.
- For remote Gems, the page server process for the remote Gem on the Stone's machine is no longer present, *and* one of the following is true:
  - The Gem's remote cache has shut down, *or*
  - The Stone has received a disconnect on the Gem's OOB socket.

If the logout is not completed after the 60-second timeout has expired, the Stone begins to kill processes.

For local Gems:

The Stone tells the Page Manager to send SIGUSR1 (kill -USR1) and SIGTERM (kill -TERM) to the non-responsive Gem process. All processes that receive the signal should immediately exit cleanly. The kill signal is only sent once. The SIGUSR1 signal is sent first to cause the Gem to print its C stack to the log file, to help you determine why the Gem was stuck.

For remote Gems:

The Page Manager sends these signals to the Gem's local page server process (if it still exists), and the Stone waits 15 seconds before checking the session again.

If the Gem's page server does not exist, and the Gem is not protected against termination (as discussed below), the Page Manager tells the page server on the *remote* host to kill the Gem process.

If the Gem is protected against termination, the Stone merely periodically checks (via the Page Manager and remote cache page server) to see if it received the OOB socket disconnect or if the remote Gem process no longer exists.

Terminating a hung Gem

If a Gem is hung, it cannot respond to either the SigAbort or SigLostOTRoot.

To clean up the Gem, the last fallback is the STN_GEM_TIMEOUT configuration parameter (by default, set to 0, disabling the timeout). This is the number of minutes after which lack of Gem interaction causes the Stone to forcibly log out the errant Gem.

A related configuration parameter, GEM_RPCGCI_TIMEOUT, lets you control the number of minutes after which lack of communication between a remote Gem and your application causes the Gem to terminate.

If all else fails, you can send a SIGTERM (kill -TERM) signal to the non-responsive session, so that it detaches from the shared cache safely and then terminates.

NOTE
Do not use SIGKILL (kill -9) to kill GemStone processes. Attempting to do so can result in repository crashes due to stuck spin locks.

For detailed information about all these configuration parameters, see Appendix A.

Protecting individual Gem sessions against termination

There are circumstances in which you might want to protect a long-running Gem session against forcible termination. You can prevent Stone from killing Gem processes (but not page server processes) on a per-session basis. To do so, execute the following method at login time:

```
System >> disableTerminationSignals
```

This method prevents Stone from sending SIGTERM (kill -TERM) to the designated session. You might use this method to preserve background Gems that *should* continue running, so that you can issue kill non-responsive user sessions as needed.

The main recommended usage of this method is with a linked Smalltalk image or a Topaz login that also has RPC sessions. If the linked session were to become non-responsive and the Stone killed it (via SIGTERM or kill -TERM), that action would also (without the benefit of this method) kill the RPC sessions, thus terminating the entire application. With this method, the RPC sessions are protected against such termination.

Because this method grants Gems immunity against kill signals, unkillable "zombie" gems could accumulate on the system. Such Gems do not cause a commit record backlog, because their view is retracted during the first phase of the logout (see "Logging out a non-responsive Gem" on page 359). However, their allocated OOPs and pages are not freed until the Stone determines it is safe to recycle these resources or the system shuts down.

Commit record backlog: side effects

The Stone is the process responsible for removing commit records from the shared page cache. While it is doing so, it is not processing commits from Gems trying to commit, thereby causing the commit queue to grow and commit performance to deteriorate for all Gems.

When a commit record causes a large backlog, other problems ensue:

- *Shadow pages*—pages holding the old, unmodified state of modified objects—cannot be listed as needing reclaiming.
- Voting cannot complete, thus impeding garbage collection; all logged-in Gems must be committing or aborting for memory to be reclaimed. (For details of garbage collection, see Chapter 10, "Managing Growth.")
- Even though pages can be reclaimed, the old pages cannot be disposed of while commit records still in the queue might refer to shadow objects on those pages.

- As the commit record backlog grows, so does the amount of work necessary for the session with the oldest commit record to compute the read/write set union to determine commit conflicts.
- Finally, if commit records fill the shared page cache, they are written to disk, making the repository larger. Temporary objects may also be written to disk, from where it is much more difficult to reclaim the pages they occupy and the object identifiers they use.

Evaluation

If **CommitRecordCount** is greater than twice STN_MAX_SESSIONS, or if the commit record count is continually growing, application performance is probably impacted by page swapping. Even if CommitRecordCount is lower than this maximum, swapping could still be a problem, especially if FreeFrameCount is also low.

To find the session or sessions connected to the oldest commit record, evaluate:

```
System sessionsReferencingOldestCr
```

TotalCommits indicates the transaction load and how fast the commit record backlog can grow.

CommitQueueSize indicates that the Stone is busy, most likely serving the commit requests of other Gems. A large commit queue size means that the cost of doing commits—of reading the read/write set union for all logged-in Gems in order to determine commit conflicts—is also growing.

Additional helpful statistics:

- **SigAbortsSent** reports the number of times the Stone has signaled a given Gem to abort, although the session may be in a sleep or I/O wait state and not yet aware of having received the signal.
- **SigAbortsReceived** reports the number of times the Stone has signaled a given Gem to abort, that it has received and recognized.
- **LostOtsSent** is the number of Lost OT Root signals the Stone has sent to a given Gem, although the session may be in a sleep or I/O wait state and not yet aware of having received the signal.
- **LostOtsReceived** is the number of Lost OT Root signals received and recognized by a given Gem.
- **CommitCount** and **AbortCount** are the number of times a Gem has committed or aborted a transaction, respectively.

- **TransactionLevel** allows you to determine the transaction status of a Gem:

  1 = in a transaction
  0 = not in a transaction
  –1 = in transactionless mode

  NOTE
  If statmonitor samples are too infrequent, you may miss transitions between levels.

Solutions

If appropriate, configure your system with a short value for STN_GEM_LOSTOT_TIMEOUT, so that unresponsive Gems can be terminated quickly. For a discussion of the trade-offs involved, see "STN_GEM_LOSTOT_TIMEOUT" on page A-418.

Alternatively, if appropriate, stop the session(s) connected to the oldest commit record by evaluating:

```
System stopSession: aSessionId
```

You can also lower the value of STN_SIGNAL_ABORT_CR_BACKLOG so that the Stone does not permit the commit record backlog to grow as large.

Another option, recommended for production systems, is to set the configuration parameter STN_GEM_TIMEOUT to automatically log off unresponsive sessions.

NOTE
A Gem running a long query will not time out and be logged off if it is getting pages and object IDs from the Stone. Requesting locks will also count as activity and will not invoke the timeout.

Make sure Gems ordinarily work in manual transaction mode and keep transactions short. You can also redesign your application so that sleeping Gems do so in transactionless mode.

When a Gem is sleeping, it will not respond to signals. Therefore, it's a good idea to program your `sleep` method in a loop, so that the Gem sleeps for half the amount of time specified in STN_GEM_TIMEOUT, then awakens briefly before returning to sleep. The interpreter activity will detect a signal, if any has been sent.

Finally, for a temporary work-around, you can increase the value of STN_MAX_SESSIONS, thus allowing the system to better tolerate the problem while you work on a real solution.

Swamped Page Server

Statistic	Default Filter	Process	Explanation
AioDirtyCount	per second	page server	See page 240.
AioCkptCount	per second	page server	See page 240.
LocalDirtyPageCount	per second	shared page cache monitor	See page 252.

Background

The asynchronous I/O page server (also called the AIO page server and hereafter called simply *page server*) is the process responsible for writing *dirty pages*—pages with modified objects—from the shared page cache to the repository extents. The Stone starts one or more page servers as part of its own startup (the number depends on the value of the STN_NUM_LOCAL_AIO_SERVERS configuration parameter). Thereafter, the page server scans the shared page cache for dirty pages and writes them to disk asynchronously, freeing the Gem from having to perform this I/O-intensive task.

At a checkpoint, the page server writes all the dirty pages from the shared page cache to the repository. In full logging mode, checkpoints occur as specified by the Stone's configuration parameter STN_CHECKPOINT_INTERVAL. In partial logging mode, a checkpoint can occur more often, if the size of a transaction exceeds the value set by the configuration parameter STN_TRAN_LOG_LIMIT.

Checkpoints are most often triggered by a Gem committing a transaction; they also occur at the start of a new transaction log or repository backup. Checkpoints may take seconds or minutes, but a checkpoint in progress does not block the system: transactions can commit as usual. In addition to all dirty pages being written to the extents, checkpoint records are written to each extent and to the transaction log.

If the page server cannot write dirty pages to disk fast enough to keep up with the Gems committing transactions, many other system processes will be delayed for lack of free frames in the shared page cache. Also, the page server will have more work to perform at each checkpoint, which can slow other processes such as page reclamation.

Evaluation

The statistics in this template help you determine if the page server can keep up with the demands created by Gems committing transactions. Compare **AioDirtyCount** to **AioCkptCount**: if a great many more pages are written at checkpoints (AioCkptCount**)** than otherwise (AioDirtyCount), the page server is probably falling behind and having to catch up at each checkpoint.

If both numbers are high, the page server could be operating at maximum.

If **LocalDirtyPageCount** is high and drops only slowly, if at all, the page server is probably not operating at peak efficiency.

Page reclamation worsens the problem: a slow page server means too few free frames in the shared page cache for the GcGem to use for copying live objects. Thus, it will spend more time scanning the cache and possibly swapping to disk.

Additional helpful statistics:

- **GlobalDirtyPageCount** (shared page cache monitor only) reports the number of dirty pages in the shared page cache that are dirty but not yet eligible for writing to disk because they're not yet committed. if this value is very large, then very large transactions may be filling the cache. If the Stone is also using this shared page cache, another possibility is that the Stone's private page cache is too small.
- **CheckpointCount** (Stone only) reports the number of checkpoints since the Stone was started. If partial logging is in effect, a rapidly increasing CheckpointCount indicates that STN_TRAN_LOG_LIMIT may be set too small.

Solutions

If the page server cannot keep up with demand, the obvious remedy is to increase the number of page servers; however, whether you realize any benefits depends on other characteristics of your system. Parallelization is of no use without parallel resources; multiple page servers boost performance only for systems with:

- over four extents (one page server can ordinarily handle up to four extents),
- multiple CPUs (to allow parallel execution), and
- extents on separate spindles or the equivalent (to allow concurrent writes to disk).

In any case, over four page servers are unlikely to provide additional benefit, unless your repository is particularly large.

Other options exist for increasing system throughput:

- Ensure that the extents, the system swap file, and the transaction logs are all on separate disk spindles.
- Consider using raw partitions if you are not already doing so.

- Experiment with non-RAID devices, if appropriate, to see if they improve performance.
- If you have spare CPU cycles, increase the I/O rate of the page server.
- Consider redesigning your application to make more efficient use of I/O operations. For example, traversing sequentially through an OrderedCollection or Array can cause an application to perform a large number of I/O operations if the referenced objects are not already in the shared page cache.
- If GlobalDirtyPageCount is high, redesign your application to use smaller transactions, if possible.
- If the shared page cache in question is used by the Stone and GlobalDirtyPageCount is high, it may help to increase the size of the Stone's private page cache.
- If partial logging is in effect and CheckpointCount is rapidly increasing, it may help to increase the value of the Stone's STN_TRAN_LOG_LIMIT configuration parameter.

Inefficient Epoch Sweep

Statistic	Default Filter	Process	Explanation
EpochGcCount	per second	Stone	See page 246.
PossibleDeadSize		Stone	See page 262.
ProgressCount		GcGem	See page 263.

Background

Garbage collection is the reclamation of pages on disk and object identifiers. The first phase of garbage collection, called *mark/sweep,* identifies objects that might be dead.

Epoch garbage collection identifies the possibly dead objects from a finite set of recent transactions, called the *epoch,* instead of the entire repository. It checks all objects created from the start time to the end time of the epoch, looking for objects that have been dereferenced. Epoch garbage collection is efficient because, for many applications, the vast majority of objects are short-lived, created to service temporary application needs and not intended to persist in the database. This is especially true of applications in which numerous small transactions mostly update a few previously committed objects.

NOTE
For a more complete description of garbage collection, see Chapter 10, especially the section entitled "Epoch Garbage Collection" on page 316.

Epoch garbage collection supplements `markForCollection`; it does not replace it. It does not find:

- objects created before the beginning of the epoch that have been dereferenced during the epoch, nor
- objects created during the epoch and not dereferenced until after the epoch has ended.

Mark/sweep actually encompasses two operations on the set of all objects created during the epoch:

1. The *mark* phase identifies all *live objects*—objects that can be reached through references starting with the AllUsers root object.
2. The *sweep* phase identifies *possibly dead objects*—all objects created in the epoch that were not identified as live objects in the first sweep, minus any unused object identifiers.

The reason these objects are considered "possibly" instead of "definitely" dead is that mark/sweep can take a significant period of time, during which a Gem may somehow commit one of the objects previously identified as possibly dead. These operations use considerable CPU cycles and disk I/O: to identify live objects, all pages containing objects created during the epoch are copied from disk into the shared page cache, if they are not already there. And the object table must be swept for unused object identifiers, so a great many object table pages are read into the GcGem's private page cache. Either or both of these operations can cause swapping, and hence be time-consuming.

Epoch garbage collection is one of several responsibilities of the GcGem. While it is running, the GcGem is not available to perform its other functions.

You can disable epoch garbage collection or change how often it runs. Assuming it's enabled, these GcGem configuration parameters control the length of an epoch:

- epochGcTimeLimit
  The maximum frequency of epoch garbage collection—by default, 15 minutes.

- epochGcTransLimit
  The number of transactions required to trigger epoch garbage collection—by default, 5000.
- epochGcByteLimit
  The number of bytes of new or modified committed objects required to trigger epoch garbage collection—by default, 5 million.
- deferEpochReclaimThreshold
  The number of pages needing to be reclaimed (another responsibility of the GcGem's) that will cause the GcGem to defer epoch garbage collection in favor of reclaiming pages—by default, 1000.

Epoch garbage collection occurs when:

```
(pages that need reclaiming < deferEpochReclaimThreshold) AND
(the time since last epoch > epochGcTimeLimit ) AND
( (transactions since last epoch > epochGcTransLimit) OR
(bytes committed since last epoch > epochGcByteLimit ) )
```

Evaluation

EpochGcCount per second should reveal that the GcGem is performing epoch garbage collection regularly, not deferring it because of a backlog of pages needing to be reclaimed.

PossibleDeadSize should also show a fairly regular graph. It should reveal that each epoch garbage collection is finding enough possibly dead objects to make it worth doing.

ProgressCount should show a fairly regular graph as well. During epoch garbage collection, ProgressCount increases as GcGem marks and sweeps the objects created during the epoch. First, it shows the number of objects marked. It then goes to zero, then up again as it shows the number of objects identified as possibly dead.

If too many objects must be swept, performance may slow because a large number of pages may have to be read into the shared page cache.

Additional helpful statistics:

- **PagesNeedReclaimSize** reports the approximate number of pages that need to be reclaimed. This statistic is updated only every 15 seconds, and includes only pages on the Stone's list. It is always an overestimate, and larger values are less accurate than smaller values.
- **GcDeferEpochThreshold** is the value of the GcGem's configuration parameter deferEpochReclaimThreshold.

Solutions

Problems caused by epoch garbage collection are typically either:

- the epoch is the wrong length (too frequent or too infrequent), or
- the application is creating too many short-lived objects.

You can tune the GcGem parameters to change the epoch length. The default period of 15 minutes tends to work better for applications that commit one or more transactions per second. To be helpful, epochs should last longer than the average lifetime of short-lived objects; for details, see “Epoch Garbage Collection” on page 316. In general, the longer the epoch, the greater the need for additional storage during the epoch. Also, bursts of garbage collection activity will be less frequent but longer. However, you’ll be able to run `markForCollection` less frequently.

If statistics reveal that few objects are garbage-collected, consider disabling epoch garbage collection or running it less frequently, when relatively few users are on the system.

Another approach is to monitor system usage and adjust epoch garbage collection to match—for example, by running it on shift boundaries, if users work in shifts.

Instructions for changing epoch length and other GcGem parameters while the system is running are provided in “Epoch Garbage Collection” on page 316. You can also change these parameters in a configuration file specific to the GcGem, so that the changes remain in effect if the system is stopped and restarted. Instructions for doing so are provided in “The GcGem” on page 398.

If PossibleDeadSize and slow performance indicate too many short-lived objects are being read during the sweep, you can:

- enlarge the shared page cache to handle the extra pages,
- shorten the epoch length, or
- redesign your application to create less garbage.

If application developers haven’t explicitly addressed the issue of garbage creation, it’s not unusual for to discover that the application puts unnecessary demands on the garbage collection.

To see what kinds of objects are created when executing a GemStone method, use ProfMonitor with object tracing turned on.

Overloaded GcGem

Statistic	Default Filter	Process	Explanation
PagesNeedReclaimSize		Stone	See page 260.
EpochGcCount	per second	Stone	See page 246.
CommitCount	per second	GcGem	See page 244.

Background

In addition to epoch garbage collection, the GcGem has several responsibilities:

- It finalizes voting among the Gems during repository-wide garbage collection.
- It reads the list of possible dead objects looking for special cases: indexed collections and compiled methods, both of which require extra processing.
- It reclaims pages with shadow objects and dead objects.

These tasks all must contend for the GcGem; see the previous discussion starting on page 366 for details, and Chapter 10 for an in-depth discussion. However, of them all, page reclamation takes longest—that's why GemStone 6.6 provides GcGems specialized for page reclaim (page 335.)

When a page is reclaimed, all the still-living objects on that page are copied to a new page, topped off with other live objects from other pages until nothing is left but shadow objects, dead objects, or free space. This is an expensive operation, particularly in terms of disk I/O: for example, the GcGem typically copies most of the object table into its private page cache, so that it can reclaim object identifiers. During page reclaim, the GcGem performs a great many pages reads from the repository.

The GcGem places the new pages in its private page cache and therefore needs a high number of cache frames. The GcGem moves new pages into the shared page cache when it completes a batch and commits the transaction.

Evaluation

PagesNeedReclaimSize reports the approximate number of pages that need to be reclaimed. This statistic is updated only every 15 seconds, and includes only pages on the Stone's list.

EpochGcCount per second should reveal that the GcGem is performing epoch garbage collection regularly, not deferring it because of a backlog of pages needing to be reclaimed.

The GcGem's **CommitCount** per second should reveal that it is able to process batches of pages and commit the new pages, thus permitting the Stone to dispose of old commit records and return pages and object identifiers to their respective free pools.

Additional helpful statistics:

- **ReclaimCount** reports the number of times the GcGem has reclaimed pages since the Stone was started.
- **ReclaimedPagesCount** reports the number of pages the GcGem has reclaimed since the Stone was started.
- **GcReclaimMaxPages** reports the current value of the GcGem's reclaimMaxPages parameter—by default, 200. This means that the GcGem will reclaim 200 empty, shadow, or dead pages in a batch before making itself available for other work, if any.
- As in the discussion of "Shared Page Cache Too Small" on page 354, the ratio of **FramesFromFindFree** to **FramesFromFreeList** can tell you if the GcGem has to spend too much time scanning the shared page cache for free frames.

Solutions

If you find that the GcGem is not keeping up with page reclamation, consider using one of the GcGems specialized for reclaiming pages; descriptions and instructions are available in "GcGems Specialized to Reclaim Pages" on page 335. To run one or more of these GcGems, you will have to temporarily halt the generic GcGem, which you can then restart after the page-reclaim backlog has been resolved. If this is an ongoing problem, you may wish to regularly schedule one or more page-reclaim GcGems.

If these specialized GcGems are impractical for you, consider increasing the GcGem's private page cache from its default size—a larger private page cache can hold more object table pages without disk swapping. Consider increasing the GcGem's private page cache to at least 20 MB. Use the procedure described starting on page 369 to increase this value in a customized configuration file.

(You may also wish to change the GcGem's GEM_FREE_FRAME_LIMIT parameter as well. For details, see the discussion beginning on page 354.)

You may also wish to tune the reclaim parameters—for example, the maximum and minimum batch sizes, to make batches either smaller (so that the GcGem isn't busy as long) or larger (so that the GcGem can get more pages reclaimed at once), depending on whether the statistics reveal a backlog of pages that need reclaiming.

Finally, if the ratio of **FramesFromFindFree** to **FramesFromFreeList** indicates that the GcGem has to spend too much time scanning the shared page cache for free frames, consider reducing the GcGem's FreeFrameLimit (for more details, see "Shared Page Cache Too Small" on page 354), or starting additional free list page servers (for instructions, see "To Add Free List Page Servers" on page 73).

Excessive In-Gem Garbage Collection

Statistic	**Default Filter**	**Process**	**Explanation**
ScavengeCount	per second	Gem	See page 265.
TimeInScavenges	per second	Gem	See page 269.
MakeRoomInOldSpaceCount	per second	Gem	See page 254.
NotConnectedObjsSetSize	per second	Gem	See page 256.

Background

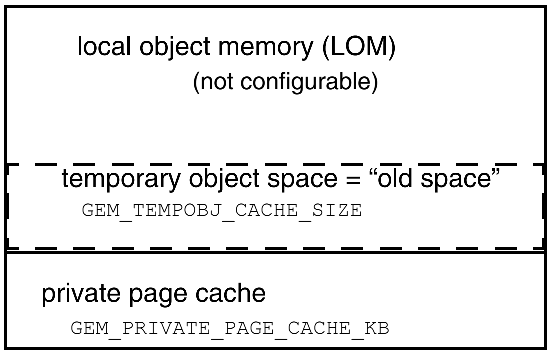
In addition to access to the shared page cache, each Gem has private memory. Much of this memory—the *local object memory*—is intended to serve as the Gem's private scratch space and is therefore organized by objects. However, some of it—the Gem's *private page cache*—is organized in 8 KB pages like the shared page cache.

To aid memory reclamation, local object memory is divided into generation spaces. For the most part, generation spaces are inaccessible to application developers and cannot be configured; however, the longest-lived generation space is a special case. Known as *temporary object space* or simply *old space,* the size of this portion of local object memory is configurable. For an object, old space is the last stop before being placed on a page.

NOTE
For additional background and basic guidelines, as well as specific tuning instructions, see "How to Tune Session Performance" on page 88.

Figure 11.1 shows the structure of a Gem's private memory:

Figure 11.1 In-Gem Memory


local object memory (LOM)
(not configurable)
temporary object space = “old space”
GEM_TEMPOBJ_CACHE_SIZE
private page cache
GEM_PRIVATE_PAGE_CACHE_KB

The Gem creates new objects smaller than 8 KB in local object memory, assigning each an object identifier. If the object is destined for persistence, it moves through local object memory generation by generation, at last ending in old space. Upon commit, the object is assigned a page in the private page cache and, from there, moves into the shared page cache.

New objects 8 KB or larger are created in the private page cache to start with.

When the Gem searches for already extant persistent objects, it searches first in its private page cache, then the shared page cache, and last in the extents. The Gem uses its private page cache to modify existing persistent objects; if the object is not already there, the Gem copies it to its private page cache and then modifies it.

The Gem performs three kinds of garbage-collection, corresponding to its three kinds of memory:

- Each generation space in local object memory is automatically garbage-collected by a process called *generation-scavenging.* A new object survives progressive generation-scavenges if it continues to be referenced.
- When old space is full, it’s garbage-collected by a special generation-scavenge called `makeRoomInOldSpace`.
- If old space is still full, temporary objects overflow into the private page cache, where they become known as *not-connected objects,* and form part of the notConnectedSet. Here they are eligible for not-connected garbage collection,

Tuning the notConnectedSet

When an application executes, it inevitably makes temporary objects. It is best if these objects are garbage-collected while still in local object memory;

complications ensue when temporaries overflow into the private page cache and become part of the notConnectedSet.

Objects become part of the notConnectedSet in one of four ways:

- Old space filled and the temporary object overflowed into the private page cache.
- The temporary object was 8 KB or larger when created and therefore began in the private page cache.
- The object was smaller than 8 KB but was referenced by a temporary larger than 8 KB. All such objects are moved into the private page cache upon a commit, where they join the large temporary in the notConnectedSet.
- A commit failed. When a Gem starts a commit operation, it moves the objects intended to become persistent into the private page cache. If the commit fails, these objects join the notConnectedSet.

The notConnectedSet is garbage-collected automatically after a generation-scavenge if the following two parameters are exceeded:

- **notConnectedThreshold**
  The minimum number of objects in the notConnectedSet required to trigger notConnectedSet garbage collection—by default, 2000.
- **notConnectedDelta**
  The minimum number of objects by which the notConnectedSet is required to grow since the last notConnectedSet garbage collection, in order to trigger the next notConnectedSet garbage collection—by default, 300.

Garbage collection of the notConnectedSet is the most expensive form of in-Gem garbage collection. The Gem has to sweep the contents of the notConnectedSet from a known root, searching for permanent references, a CPU-intensive operation. If the private page cache pages have been swapped to disk, the operation becomes I/O-intensive as well.

Objects in the notConnectedSet are uncommitted until a successful commit. Upon commit, some objects in the notConnectedSet become committed objects, eligible to be written to the extents by the page server.

If an object in the notConnectedSet is unreferenced when not-connected garbage collection runs, the Gem can remove the object from the notConnectedSet and delete it. After an object has been committed, however, the Gem cannot safely delete it. Instead, the object must be removed by one of the two forms of repository-wide garbage collection: epoch garbage collection, or `markForCollection`.

Evaluation

ScavengeCount reports the number of times generation-scavenging has run since the Gem was started. **TimeInScavenges** reports the cumulative CPU time, in milliseconds, spent in generation-scavenging. Objects collected during generation-scavenging are objects that don't enter the notConnectedSet, never become committed and written to disk, Together, these two statistics can be an indication of how efficiently your application is coping with temporaries.

If **MakeRoomInOldSpaceCount** increments frequently, local object memory may be overflowing and adding to the notConnectedSet.

NotConnectedObjsSetSize reports the size of the notConnectedSet. If its value grows consistently, temporary object space may be too small, or your application may be creating too many large temporaries. Compare the NotConnectedObjsSetSize with MakeRoomInOldSpaceCount to determine if frequent overflows from temporary object space are causing the notConnectedSet to grow.

If the **NotConnectedObjsSetSize** grows after each commit, one or more large temporaries is probably referring to other, smaller temporaries which are being pulled into the notConnectedSet upon commit. Further diagnostic tools:

- To determine if an object is committed, send it the message `isCommitted`.
- To determine if an object is connected, send it the message `isConnected`.
- To determine the size of an object in bytes, send it the message `physicalSize`.

Additional helpful statistics:

- **DeadObjCount** reports the number of dead objects collected in the Gem.
- **FailedCommitCount** is useful for determining if the Gem has suffered a great many failed attempts to commit. Such attempts could grow the notConnectedSet significantly. Compare this statistic with **CommitCount** to see the ratio of successful to failed commits.
- **GcNotConnectedCount** measures how many times the Gem has garbage-collected the notConnectedSet since the Gem was started. Compare this with NotConnectedObjsSetSize to determine if garbage collection is decreasing the notConnectedSet.
- **GcNotConnectedDeadCount** measures the total number of dead objects that the Gem found during all garbage-collections of the notConnectedSet since the Gem started. Apply a per-second filter to view how many dead objects the Gem found during the last garbage-collection cycle of the notConnectedSet.

Solutions

If **MakeRoomInOldSpaceCount** increments frequently, increasing the size of old space may improve performance.

If your application makes many new persistent objects, you may instead wish to commit more frequently. This will cause new persistent objects to move out of old space and into the shared page cache more often, thus making more room in old space.

Finally, redesign your application to make more efficient use of temporaries. For example, reuse large temporary objects such as collections. Collections are dynamically resizable in GemStone Smalltalk; instead of making a new one, resize and old one to 0 and reuse it.

If the NotConnectedObjsSetSize keeps growing, collect notConnectedSet garbage before committing. To do so, evaluate:

```
System markNotConnectedForCollection
```

This operation is also useful after a failed commit.

Or, if statistics indicate that notConnectedSet garbage-collection should happen more often on a regular basis, lower the values of the parameters **notConnectedThreshold** and **notConnectedDelta**.

11.2 Garbage Collection for Tuners

This section is for those who have identified garbage collection as a serious performance bottleneck. Below is a capsule summary of a long and complex process: for complete details, see Chapter 10, “Managing Growth.”

Following this is a discussion of problems you might encounter during these various steps, their symptoms, and potential remedies. Following that discussion are three examples on an actual database with default and new values for modified parameters.

You’ll get more out of this section if you take the time now to consider these questions:

- What problems are you experiencing? Host problems such as CPU availability, I/O bandwidth, disk swapping, network bandwidth are best diagnosed with host operating system tools. When you have characterized the problem to this level, it can be helpful to look further.

- What system behavior is causing the problem? Slow commits? Repository growth? System login delays?
- In tuning garbage collection, what is your goal? Do you wish it to run as fast as possible, or to disturb other users as little as possible? Or perhaps you'd like a balance between these two goals?

Example Garbage Collection Cycle

GemStone/S has several kinds of garbage collection: the automatically run epoch garbage collection and in-Gem garbage collection of several kinds, as well as the explicitly run `markForCollection`, `markGcCandidates`, and various kinds of `startGC:`. This particular example starts with a run of `markForCollection`:

1. The Gem running `markForCollection` finds all the live objects by traversing references, starting at the system root AllUsers.
2. It computes the *set of possible dead objects* as follows:
   — Subtract the live objects from the universe of possible objects: objects whose IDs range from zero to the highest object ID in the system.
   — Also, subtract all the unassigned object IDs in that range.

   The object table is a key data structure for this step.
3. The Gem running `markForCollection` sends the Stone the list of possibly dead and returns.

   The Stone now holds the possible dead set in RAM until the next checkpoint. If the system should fail at this point, you'll have to run `markForCollection` again.

   NOTE
   Discussion of the rest of these steps applies also to epoch and special-candidate garbage collection.
4. Now every Gem currently logged in the system must search the possible dead set for any objects to which it holds references. Then it must commit or abort, at which time it votes to either keep an object in the set, or remove it (if it holds a reference).

   Here's where Gems that sleep in transactions cause trouble. Without committing or aborting, they do not vote. The vote cannot be finalized, garbage collection halts at this point, and commit records accumulate.
5. But what about Gems that aren't on the system now, but were when garbage collection started? Their modified objects are in the commit record backlog, in

the write sets of each commit record, which the GcGem reads in order to vote on their behalf.

While doing so, the GcGem is unavailable for its other functions.

6. The resulting set now holds nothing but unreferenced objects. If some of those objects are compiled methods or indexed collections, however, additional work must be done. The GcGem now hunts through the set of possible dead looking for those special cases.
7. The resulting objects are now deemed dead.
8. Pages are reclaimed. Repository shrinkage is now visible.
9. Page identifiers and object identifiers are returned to the free pools.

Step-by-Step Tuning

Each step in garbage collection presents a different opportunity for optimization.

Step 1. Identify live objects.

Step 1, identifying all the live objects—the *mark* phase—is where `markForCollection` takes most of its time.

Factors affecting duration

size of `markForCollection` Gem's mark/sweep buffer	larger is faster
number of objects in repository	fewer is faster
maximum disk I/O speed	faster is faster (surprise!)
shared page cache size	larger is faster
ratio of object table size to shared page cache size	faster if entire object table fits in shared page cache
number of other Gems sharing the shared page cache	fewer is faster
`markForCollection` Gem's private page cache size	larger is faster
`markForCollection` Gem's GEM_IO_LIMIT	larger is faster

Statistics

ProgressCount
During this step, progress count displays the number of objects marked as live. The final value, at this step, is the total number of live objects.

Tunable parameters

size of `markForCollection` Gem's mark/sweep buffer	increase this
shared page cache size	larger is faster
ratio of object table size to shared page cache size	faster if entire object table fits in shared page cache
`markForCollection` Gem's private page cache size	increase this
`markForCollection` Gem's GEM_IO_LIMIT	depends on system goals
System's STN_GEM_ABORT_TIMEOUT	increase this if you increase GEM_IO_LIMIT
System's DBF_ALLOCATION_MODE	equally weighted for multiple extents

Step 2. Compute possible dead set.

Factors affecting duration

number of possible dead objects in repository	fewer is faster
the largest object identifier, known as the *OOP high water mark*	lower is faster
ratio of object table size to shared page cache size	faster if entire object table fits in shared page cache
`markForCollection` Gem's GEM_IO_LIMIT	larger is faster

Statistics

ProgressCount
During this step, progress count displays the number of objects identified as possibly dead. The final value, at this step, is the total number of possibly dead objects.

Tunable parameters

shared page cache size
This step goes faster if the entire object table fits into the shared page cache. To compute the approximate size of the object table, evaluate:

```
(System _oopHighWaterMark // 4) * 6
```

Step 3. Return possible dead set to Stone.

This step happens quickly and needs no tuning.

Statistics

PossibleDeadSize
The Stone's set of possible dead objects. This value is only approximate.

Step 4. Logged-in Gems vote.

Factors affecting duration

average transaction length	shorter is faster
the size of the possible dead set	smaller is faster
number of Gems logged in during vote	fewer is faster
network bandwidth between Stone's host and remote Gems' host	more is faster
number of objects in notConnectedSet written to disk	fewer is faster

Statistics

VoteNotDead
The number of objects in the possible dead set for which a given Gem holds a reference.

Tunable parameters

Transaction length—voting completes faster if Gems run short transactions, as they vote upon commit or abort.

Step 5. GcGem votes for logged-out Gems.

Factors affecting duration

number of objects in the commit record backlog	fewer is faster
the size of the possible dead set	smaller is faster
maximum disk I/O rate	faster is faster
size of the shared page cache	larger is faster
the GcGem's GEM_IO_LIMIT	faster is faster

Statistics

ProgressCount
In this step, how far through the write-set union the GcGem has swept.
This value peaks at GcPossibleDeadWsUnionSize and falls back to 0.

GcPossibleDeadWsUnionSize
The size of the write-set union the Stone holds, which the GcGem must search on behalf of Gems no longer logged in.

GcPossibleDeadSize
The exact number of possible dead objects after voting.

GcSweepCount
The number of times this GcGem has performed this step since it was started.

Tunable parameters

None.

Step 6. GcGem hunts for special cases.

Factors affecting duration

number of dead UnorderedCollections (nonsequenceable collections) with indexes	fewer is faster
the size of the possible dead set	smaller is faster

maximum disk I/O rate	faster is faster
size of the shared page cache	larger is faster
GcGem's GEM_IO_LIMIT	faster is faster

Statistics

ProgressCount
In this step, how far through the possible dead set the GcGem has swept. This value peaks at GcPossibleDeadSize.

Tunable parameters

- Shared page cache size
  This step goes faster if the entire object table fits into the shared page cache. To compute the approximate size of the object table, evaluate:

  ```
  (System _oopHighWaterMark // 4) * 6
  ```

- The GcGem's GEM_IO_LIMIT
  Reduce this only if evaluation reveals that the GcGem is degrading system-wide performance due to disk I/O. Other functions of the GcGem will also be affected.

Step 7. The possibly dead are now dead.

This step happens quickly and needs no tuning.

Statistics

PossibleDeadSize and **GcPossibleDeadSize**
These values fall to zero during this step.

DeadNotReclaimedSize
The number of objects finally in the possible dead set.

Step 8. Pages and object IDs are reclaimed.

This can be the most time-consuming step. Furthermore, the GcGem performs this while in a transaction, because reclaimed pages and object identifiers must be committed.

Factors affecting duration

presence and number of dedicated page-reclaim GcGems	Though they cannot be run with the generic GcGem, scheduling dedicated page-reclaim GcGems to run intermittently can significantly reduce the number of pages needing to be reclaimed,
GcGem's reclaimMaxPages and reclaimMinPages settings	Controls number of pages reclaimed per commit; larger batches mean longer transactions, more pages reclaimed, less other work done.
GcGem's epochGcEnabled setting GcGem's epochTimeLimit setting	If the GcGem has to spend time running epochs, it can spend less time reclaiming pages.
GcGem's FREE_FRAME_LIMIT	lower is faster
GcGem's GEM_IO_LIMIT	faster is faster
GcGem's private page cache size	larger is faster
size of the shared page cache	larger is faster
ratio of object table size to shared page cache size	faster if entire object table fits in shared page cache
number of shadow objects	GcGem reclaims shadow pages before reclaiming pages with dead objects.
number of free frames on free frame list	more is faster
size of commit record backlog	smaller is faster

Statistics

DeadNotReclaimedSize
The number of objects in the Stone's list of objects ready to be reclaimed. This value should decrease as dead objects are reclaimed.

PagesNotReclaimedSize
The number of pages in the Stone's list of pages ready to be reclaimed. This value should decrease as pages with dead objects are reclaimed.

GcReclaimNewDataPagesCount
The number of pages the GcGem has reclaimed so far during this step.

FreeFrameCount
The number of free frames on the free frame list. This value should increase.

Tunable parameters

- The number of dedicated page-reclaim GcGems
  Consider scheduling these regularly if page reclamation is consistently behind. See "GcGems Specialized to Reclaim Pages" on page 335 for details.
- STN_DEAD_X_LOCKING_ENABLED
  This system configuration parameter is by default set to true, meaning that all objects identified as dead in the previous step are added to the Stone's exclusive lock set. This prevents any other Gem from committing a transaction that modifies them. If the set is large—for example, tens of millions of objects—this can degrade performance for all Gems, as they must search the set before committing any transaction.

  If you trace a performance problem to a large number of deadNotReclaimed objects, and you are certain that your application never locates objects directly by object identifier, then set this parameter to false and see if performance improves.

  CAUTION
  With STN_DEAD_X_LOCKING_ENABLED *set to false, it becomes possible to commit a reference to a dead object, thus corrupting your database. Modify this setting* only if you are sure *your application never locates objects directly by object identifier.*

- GcGem's reclaimMaxPages setting (in GcUser's UserGlobals)
  The GcGem commits its page reclamation transaction when it has reclaimed reclaimMaxPages, or when no more pages need reclaiming. The GcGem must perform a lot of work before the transaction can be committed; therefore, large values can slow the system for other Gems. However, a small value could

mean that the GcGem falls behind in page reclamation and the repository grows. Typical values for large databases are between 500 and 3000.

- GcGem's GEM_FREE_FRAME_LIMIT (in GcGem's configuration file)
  Gems have free frame limits to ensure that the Stone never runs out of free frames, because if it does, performance suffers throughout the system. If evaluation indicates that performance is slow because the GcGem has to scan the cache for free frames, consider adjusting this value downward cautiously, monitoring to ensure that the free frame list never falls to zero.
- GcGem's epochGcEnabled and epochTimeLimit settings
  If page reclamation is falling behind, consider temporarily disabling epoch garbage collection, or running it less often.
- The GcGem's GEM_IO_LIMIT
  Consider loosening the limit, or giving the GcGem access to unlimited I/O operations, if page reclaim is falling behind. You may wish to tighten the limit if page reclamation is slowing other Gem's unacceptably.
- The size of the GcGem's private page cache
  Consider enlarging it if the GcGem is spending a lot of time hunting for free frames.

Step 9. Page IDs and object IDs are returned to free pools.

For the most part, this step happens quickly and needs no tuning; however, difficulties can occur if a large commit record backlog builds up.

Statistics

DeadObjsCount
The number of object identifiers returned to the free pool since the Stone was started. This value should increase.

FreePages
The number of free pages in the repository. This value increases when the page reclaim commit record is disposed of.

ReclaimCount
The number of page reclaim transactions the GcGem has performed since it was started.

ReclaimedPagesCount
The number of pages the GcGem has reclaimed since it was started. This value should increase.

Three Examples

Three examples below tune a database for:

- the fastest garbage collection,
- the least disruptive garbage collection, and
- the best compromise between these two somewhat conflicting goals.

All refer to the system described in Table 11.1:

Table 11.1 Example Database for Garbage Collection Tuning

GemStone version:	6.6
OOP high water mark:	300,000,000
CPUs:	4
RAM (physical):	2 GB
repository size:	9.5 GB
disk drives:	6
extents:	6, one per dedicated disk even extent allocation
maximum extent:	2 GB
shared page cache:	768,000 KB 750 MB 96,000 pages
average free frames:	4800
minimum free frames:	3,000
free frame limit (Gems)	default: 9600
free frame limit (GcGem)	default: 4800 **new: 2160 = 120% of (average – minimum)**
object table:	450 MB
object table / shared page cache:	1.667

NOTE
All three examples use a free frame limit for the GcGem that is lower than the default.

LEGEND
mfcGem = the Gem running `markForCollection`.

Example 1. Faster

The following example tunes the example database for the fastest possible garbage collection cycle.

Table 11.2 Tuned for Fast Garbage Collection

Parameter	Where to change it	Setting	Comments
`#mfcGcPageBufSize`	mfcGem UserGlobals	3000	Increase size of mark/sweep buffer
GEM_PRIVATE_PAGE_CACHE_KB	mfcGem config file	65536	maximum
GEM_IO_LIMIT	mfcGem config file	5000	limited only by host file system performance
STN_GEM_ABORT_TIMEOUT	system config or runtime	15	Increase to long enough to accommodate `markForCollection`.
GEM_IO_LIMIT	GcUser UserGlobals	5000	limited only by host file system performance
epochGcEnabled	GcUser UserGlobals	false	Disable epochs while running `markForCollection`
reclaimMaxPages	GcUser UserGlobals	1000	Increase to reclaim more pages.
GEM_PRIVATE_PAGE_CACHE_KB	GcGem config file	65535	Increase to make page reclaim go faster.
GEM_FREE_FRAME_LIMIT	GcGem config file	2160	Decrease to make page reclaim go faster.

Example 2. Nicer

The following example tunes the example database for the least possible disruption to other system users.

Table 11.3 Tuned for Least Disruptive Garbage Collection

Parameter	Where to change it	Setting	Comments
`#mfcGcPageBufSize`	mfcGem UserGlobals	3000	Increase size of mark/sweep buffer
GEM_PRIVATE_PAGE_CACHE_KB	mfcGem config file	65536	maximum
GEM_IO_LIMIT	mfcGem config file	50	Limit the number of I/O operations per second so that other users can access the file system.
STN_GEM_ABORT_TIMEOUT	system config or runtime	60	Increase to compensate for low GEM_IO_LIMIT.
GEM_IO_LIMIT	GcUser UserGlobals	100	Limit the number of I/O operations per second so that other users can access the file system.
epochGcEnabled	GcUser UserGlobals	true	Default. Allow epochs to run as usual.
reclaimMaxPages	GcUser UserGlobals	300	Decrease due to low I/O limit.
GEM_PRIVATE_PAGE_CACHE_KB	GcGem config file	200	default
GEM_FREE_FRAME_LIMIT	System config file	2160	Allow other Gems to take more free frames from free frame list.
FREE_FRAME_LIMIT	GcGem config file	9600	GcGem must scan cache for free frames more often than other Gems.

Example 3. Fast enough, nice enough

The following example is a compromise between the goals of fast garbage collection and minimal disruption to system users.

Table 11.4 Compromise Tuning for Garbage Collection

Parameter	Where to change it	Setting	Comments
`#mfcGcPageBufSize`	mfcGem UserGlobals	3000	Increase size of mark/sweep buffer
GEM_PRIVATE_PAGE_CACHE_KB	mfcGem config file	65536	maximum
GEM_IO_LIMIT	mfcGem config file	5000	limited only by host file system performance
STN_GEM_ABORT_TIMEOUT	system config or runtime	15	Increase to long enough to accommodate `markForCollection`.
GEM_IO_LIMIT	GcUser UserGlobals	5000	limited only by host file system performance
epochGcEnabled	GcUser UserGlobals	true	Default. Allow epochs to run as usual.
reclaimMaxPages	GcUser UserGlobals	500	Intermediate value is a compromise between page reclaim and other Gems' work.
GEM_PRIVATE_PAGE_CACHE_KB	GcGem config file	65535	Increase to make page reclaim go faster.
GEM_FREE_FRAME_LIMIT	GcGem config file	2160	Decrease to make page reclaim go faster.

Discussion

Two things are always true:

- The Gem running `markForCollection` always wants a larger mark/sweep buffer. To change it, for the Gem running `markForCollection`, evaluate:

  `UserGlobals at: #mfcGcPageBufSize put:` *newValue*

- The Gem running `markForCollection` always wants the largest possible private page cache.

These patterns emerge:

- The `markForCollection` Gem's GEM_IO_RATE can be a bottleneck. If disk waits are a problem during `markForCollection`, increase it. Allow it to perform as many I/O operations per second as necessary.
- If you give the `markForCollection` Gem an unlimited GEM_IO_RATE, increase STN_GEM_ABORT_TIMEOUT to accommodate `markForCollection`, or a SigAbort and the following ABORT_LOST_OT_ROOT errormay cause the `markForCollection` to fail, possibly wasting a great deal of time.
- The GcGem's GEM_IO_RATE can also be a bottleneck. If disk waits are a problem during voting or epoch garbage collection, increase it.
- Increasing the size of the GcGem's private page cache allows it to reclaim pages faster.
- So does decreasing the size of the GcGem's free frame limit.
- The GcGem's configuration parameter reclaimMaxPages is most sensitive to speed vs. system impact. The default is 200; in practice, for this example system, values of 300–1000 provide a wide range of behavior from fast but disruptive to users, to slower but with less impact to users.
- Disabling epoch garbage collection while running `markForCollection` is not necessary.

Appendix

A

GemStone Configuration Options

A GemStone configuration file is a file containing information that, when read at start-up 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.
- **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.

System-wide configuration files are located by a GEMSTONE_SYS_CONF environment variable, and 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, for C shell:

```
% setenv GEMSTONE_EXE_CONF $HOME/myFile.conf
```

or, for Bourne or Korn shell,

```
$ 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, gemsetup, pageaudit** and **topaz** 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

A.1 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.

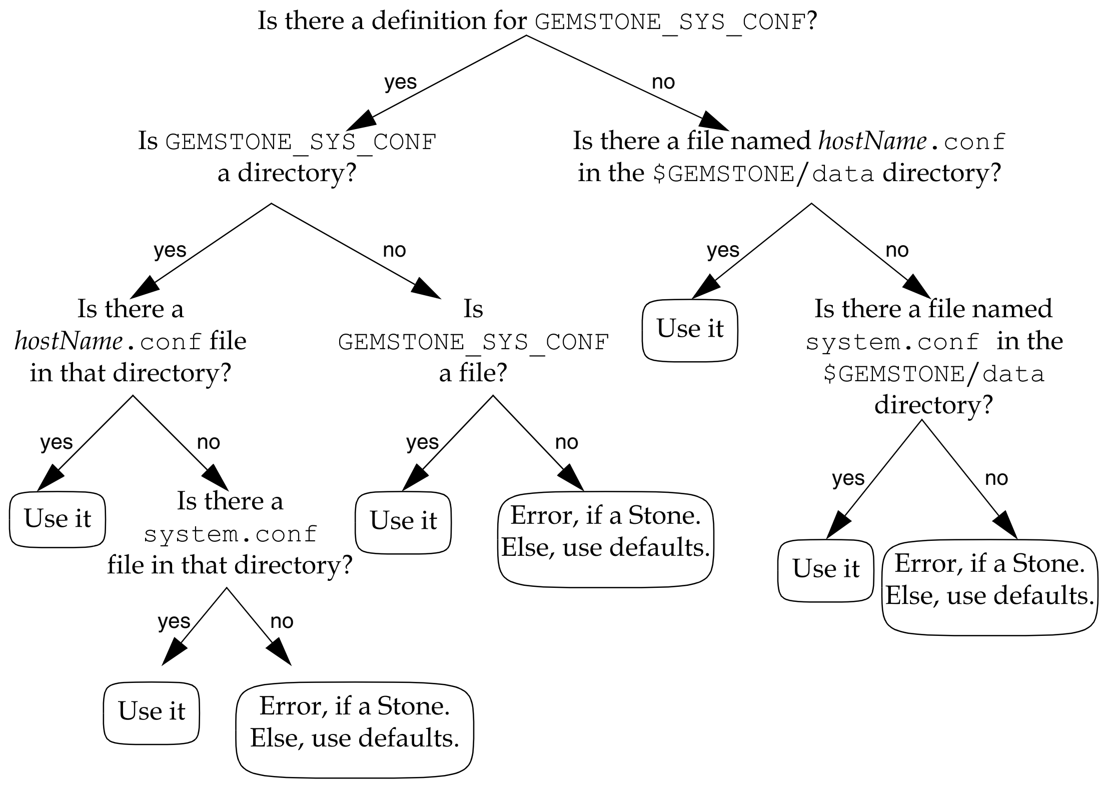
1. As shown in Figure A.1, GemStone first checks to see if there is an environment variable defined for GEMSTONE_SYS_CONF.
2. If GEMSTONE_SYS_CONF is *not* defined, GemStone looks for a file named *hostName*`.conf` in `$GEMSTONE/data` and uses that file. *hostName* must match the results of executing the `hostname` command on the machine on which the executables are running.
3. If no such file exists, it looks for a file named `system.conf` in `$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.

- 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; if not, 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 the 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

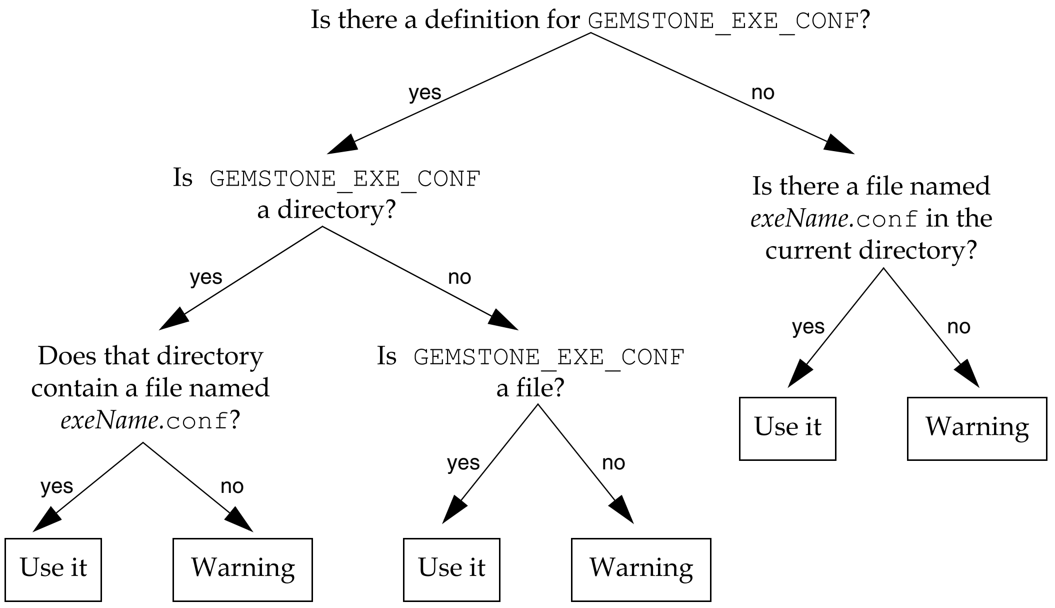

Is there a definition for GEMSTONE_SYS_CONF?
yes
no
Is GEMSTONE_SYS_CONF a directory?
Is there a file named hostName.conf in the $GEMSTONE/data directory?
yes
no
yes
no
Is there a hostName.conf file in that directory?
Is GEMSTONE_SYS_CONF a file?
Use it
Is there a file named system.conf in the $GEMSTONE/data directory?
yes
no
yes
no
Use it
Is there a system.conf file in that directory?
Use it
Error, if a Stone. Else, use defaults.
yes
no
Use it
Error, if a Stone. Else, use defaults.
yes
no
Use it
Error, if a Stone. Else, use defaults.

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 398.)
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


Is there a definition for GEMSTONE_EXE_CONF?
yes
no
Is GEMSTONE_EXE_CONF a directory?
Is there a file named exeName.conf in the current directory?
yes
no
Does that directory contain a file named exeName.conf?
Is GEMSTONE_EXE_CONF a file?
yes
no
Use it
Warning
yes
no
yes
no
Use it
Warning
Use it
Warning

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, then 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 files `$GEMSTONE/sys/gemnetobject` (Bourne shell) and `$GEMSTONE/sys/gemnetobjcsh` (C shell) are scripts that a NetLDI invokes to start a GemStone session process. These scripts can be edited to define the name of the Gem to execute, the directory where the Gem resides, and the GEMSTONE_SYS_CONF and GEMSTONE_EXE_CONF environment variables.

The GcGem

It is sometimes useful to change GcGem parameters in a configuration file specific to the GcGem, 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`. Place it in GemStone's `bin` directory.

Step 4. Edit the GcGem scripts in the `$GEMSTONE/sys` directory to use the new configuration file. Change the line:

```
exeConfig=""                        # GEMSTONE_EXE_CONF
```

to read:

```
exeConfig="$GEMSTONE/bin/gcgem.conf"
```

The GcGem scripts are `rungc` (for the stand-alone GcGem), and `rungcepc`, `rungcpdr`, `rungcpsr`, and `rungcrcl` for the specialized GcGems.

Stones

Stones look 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 GemBuilderfo C 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().

Linkable GBS Applications

Linkable GemBuilder for Smalltalk applications look for a file named `gbs.conf` in the current working directory .

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 will 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, once both the system-wide and executable-dependent configuration files have been processed, the values of all the options that the executable understands are displayed. You can access the configuration parameters from Smalltalk by using the methods described starting on page 66.*

A.2 Configuration File Syntax

The following section describes the rules of grammar to be used in editing configuration files.

White space	Leading white space 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 (#). To embed comments in a configuration file, do either of the following: ● Start a line with the comment symbol. ● Place any text after the statement termination symbol (;).
Lists	Lists are separated by commas; list elements can be empty, for example: `DBF_REPLICATE_NAMES = ,,foo.dbf;` 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:

To generate:	Use the sequence:
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, then information about the error is written to standard output. This information includes 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 three-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 will ignore the given value and use the option's default value.

NOTE
After modifying any parameter in a configuration file, and after GemStone system upgrades, check for warnings both in the stone log and in a gem log (for GEM_* configuration options). Unnoticed option value errors that result in using the default value may strongly affect application performance.

A.3 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 at the end of the entry. For more information, see "To Change Settings at Run Time" on page 67.

Note that `$GEMSTONE/bin` directory contains a write-protected file named `initial.config` that is an exact replicate of `$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.

CONCURRENCY_MODE

Internal parameter: **#ConcurrencyMode**

CONCURRENCY_MODE is used to control concurrency conflict checking.

Permissible values are FULL_CHECKS (default) and NO_RW_CHECKS.

These are defined as follows:

FULL_CHECKS	Both read/write and write/write conflicts are detected.
NO_RW_CHECKS	Only write/write conflicts are detected.

Internal settings: 0 = FULL_CHECKS, 1 = NO_RW_CHECKS.

Changing the internal parameter requires the SessionAccess privilege, and the session making the change must be the only one logged in (other than GcUser).

Default: FULL_CHECKS

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.

Default: Sequential

DBF_EXTENT_NAMES

DBF_EXTENT_NAMES is a 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 must 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 if an extent list is not defined.
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, which is 16 GB.

For optimal performance using a raw partition, DBF_EXTENT_SIZES should be slightly smaller than the size of the partition so that 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: 16384

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.

Moreover, when DBF_PRE_GROW is set to True, existing extents are affected 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.

An extent without a maximum size is not pregrown to any size; it is allocated a minimum size determined internally by the GemStone system.

Default: False (extents will grow only when new space is needed by the logical repository)

DBF_REPLICATE_NAMES

DBF_REPLICATE_NAMES lists the replicates of all repository extents, in order corresponding to DBF_EXTENT_NAMES, separated by commas. A replicate name may be omitted to indicate that the replicate is not to be used.

Default: Empty

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_ATTACHED_PAGE_LIMIT

GEM_ATTACHED_PAGE_LIMIT specifies the maximum number of pages that can be attached by a single Gem while free frames are available in the shared page cache.

The default setting of -1 causes this value to be set to 5% of the frames in the shared cache, the minimum allowed value. The maximum value allowed is 70% of the frames in the shared cache.

Out-of-range values do not generate warnings; in such cases, the default value is (silently) used.

This parameter should only be changed if the cache statistic `AttachDeltaPagesSatisfiedCount` for a given Gem is a non-zero value. Otherwise, it will have no effect.

Units: cache frames
Default: -1
Minimum: 5% of the frames in the shared page cache.
Maximum: 70% of the frames in the shared page cache.

GEM_DBF_FILE_LOCK

When GEM_DBF_FILE_LOCK is True, the Gem does advisory locking of repository extents when opening the DBF file during login.

Default: False

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 that the remaining frames in the list are available for use by the Stone repository monitor.

Default: Set at login to 10% of the actual cache size; for example, 125 frames when using the default cache size of 10 MB (1250 frames)
Minimum: 1
Maximum: 65536

GEM_FREE_PAGEIDS_CACHE

Internal parameter: **#GemFreePageIdsCache** (read-only access)

Specifies the maximum number of free pageIds to be cached in gem. Larger values reduce number of calls to stone, at the cost of needing more free space within the extents.

Default: 200
Minimum: 40
Maximum: 1000

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 (this limit also applies to linked Gems). Values greater than 10000 result in the I/O rate being limited only by the performance of the underlying file system or disk partitions.

Default: 10000
Minimum: 1
Maximum: 65536

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_NATIVE_CODE_MAX

Internal parameter: **#GemNativeCodeMax**

NOTE
Some architectures may not support native code.

GEM_NATIVE_CODE_MAX determines the maximum size of a native code method, in words. Native methods are inherently less compact than portable methods. This parameter may be useful in memory-limited environments to prevent extremely large methods from being converted to native.

The settings have these meanings:

< 0	No limit on the native code size.
0	Native code generation is disabled. Note: it is more efficient to disable native code by setting GEM_NATIVE_CODE_THRESHOLD to –1.
> 0	The maximum size (in words) of a native code method. Larger positive values permit larger methods to be converted.

Default: architecture-specific

GEM_NATIVE_CODE_THRESHOLD

Internal parameter: **#GemNativeCodeThreshold**

NOTE
Some architectures may not support native code.

GEM_NATIVE_CODE_THRESHOLD is the invocation count at which a GsMethod will be converted to native code. The settings have these meanings:

–1 Native code generation is disabled. This is the preferred way to disable native code.

0 Native code generation is always performed, even for one-time execution from a workspace.

>0 Native code generation is performed if and when the invocation count for the method exceeds the given value. Larger positive values cause less aggressive conversion.

Default: architecture-specific

GEM_NOT_CONNECTED_DELTA

Internal parameter: **#NotConnectedDelta**

GEM_NOT_CONNECTED_DELTA specifies the minimum number of objects by which the notConnectedSet is required to grow since the last notConnectedSet garbage collection, in order to trigger the next notConnectedSet garbage collection. For information, see “Collecting the NotConnectedSet” on page 316.

Default: 300

GEM_NOT_CONNECTED_THRESHOLD

Internal parameter: **#NotConnectedThreshold**

GEM_NOT_CONNECTED_THRESHOLD specifies the minimum number of objects in the notConnectedSet required to trigger garbage collection of the notConnectedSet. For information, see “Collecting the NotConnectedSet” on page 316.

Default: 2000

GEM_PGSVR_COMPRESS_PAGE_TRANSFERS

Internal parameter: **#GemPgsvrCompressPageTransfers**

If GEM_PGSVR_COMPRESS_PAGE_TRANSFERS is TRUE, use `compress2()` from zlib library with default compression level to compress page transfers between pgsvr on the Stone's machine and the Gem or mid-level cache pgsvr.

For the first gem to login on a remote machine, that Gem's configuration file value of this parameter is propagated to the Page Manager and used to configure the Page Manager's communication to Page Manager's pgsvr on the new remote cache.

When a gem triggers creation of a mid-level cache, via the method `midLevelCacheConnect:cacheSizeKB:maxSessions:`, that Gem's current runtime value of this parameter is propagated to the Page Manager and used to configure the Page Manager's communication to Page Manager's pgsvr on the new mid-level cache.

Default: False

GEM_PGSVR_FREE_FRAME_LIMIT

GEM_PGSVR_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 406).

If the value of GEM_PGSVR_FREE_FRAME_LIMIT is –1, the free frame limit is set to 10% of the shared cache size used by the page server.

To tune the free frame limit of a page server at runtime, use the method `System class>>changeCacheSlotFreeFrameLimit:` *aSlot* `to:` *aValue.*

Default: –1 (10% of cache size)
Minimum: –1
Maximum: 65536

GEM_PGSVR_UPDATE_CACHE_ON_READ

Internal parameter: **#GemPgsvrUpdateCacheOnRead**

GEM_PGSVR_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 (that is, Gems that are running on the Stone's machine).

This option has no effect on a mid-level cache. On a mid-level cache, a cache miss always updates the mid-level cache with the result obtained from reading the page from Stone's cache or disk.

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: 200
Minimum: 64
Maximum: 524288 (see note)

NOTE
The actual maximum for this parameter is dependent on the operating system memory mapping protocol and the sizes of other GemStone caches. See the note in SHR_PAGE_CACHE_SIZE_KB *(page 413) for details.*

GEM_RPC_KEEPALIVE_INTERVAL

Interval in seconds for the RPC GCI client keep-alive packet to be sent on the seldom used out-of-band (OOB) socket between the gem and the GCI client. Has no effect for linked sessions or RPC sessions running the same host as the gem process.

If enabled, keep-alive packets are sent during the GciPollForSignal() call.

Default: 0 (disabled)
Minimum: 0
Maximum: 7200

GEM_RPCGCI_TIMEOUT

GEM_RPCGCI_TIMEOUT specifies the time (in minutes) after which the 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_TEMPOBJ_CACHE_SIZE

Internal parameter: **#GemTempObjCacheSize**.

GEM_TEMPOBJ_CACHE_SIZE sets the size (in KB) of the Gem's temporary object space. (This limit also applies to linked Topaz sessions and linked GemBuilder applications). The total memory allocation (in KB) for managing temporary objects is (400 + GEM_TEMPOBJ_CACHE_SIZE) for GEM_TEMPOBJ_CACHE_SIZE ≥400, or (200 + GEM_TEMPOBJ_CACHE_SIZE) for GEM_TEMPOBJ_CACHE_SIZE <400.

If you increase the internal parameter at run time, it is most efficient (in terms of memory usage) to do so as soon as possible after logging in, preferably soon enough that no garbage collection has occurred in the temporary object space. Attempting to set the size smaller than its current size generates an error.

Default: 600
Minimum: 200
Maximum: 20000 (see note)

NOTE
The actual maximum for this parameter is dependent on the operating system memory mapping protocol and the sizes of other GemStone caches. See the note in SHR_PAGE_CACHE_SIZE_KB *for details.*

KEYFILE

KEYFILE sets the location of the GemStone licensing key file.

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 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_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 log in 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 system calculates a value for this parameter based on:
STN_MAX_SESSIONS (for user sessions)
+ number of extents in repository (for GcGems)
+ SHR_NUM_FREE_FRAME_SERVERS (for free frame page servers)
+ STN_NUM_LOCAL_AIO_SERVERS (for AIO page servers)
+ 2 (for stone and pcmon processes)

Default: -1
Minimum: number of extents (for startup page servers)

+ SHR_NUM_FREE_FRAME_SERVERS
+ STN_NUM_LOCAL_AIO_SERVERS
+ 2 (for stone and pcmon processes)

Maximum: STN_MAX_SESSIONS
+ 256 (maximum possible number of extents/GcGems)
+ 30 (maximum possible value for SHR_NUM_FREE_FRAME_SERVERS)
+ 30 (maximum possible value for STN_NUM_LOCAL_AIO_SERVERS)
+ 2 (for stone and pcmon processes)

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: 10000
Minimum: 512
Maximum: Limited by system memory, kernel configurations, and cache space allocated by SHR_PAGE_CACHE_NUM_PROCS

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.

Note that the platform-specific limitations assume default configuration sizes for the other caches (STN_PRIVATE_PAGE_CACHE_KB for the Stone, GEM_PRIVATE_PAGE_CACHE_KB and GEM_TEMPOBJ_CACHE_SIZE for the Gem). As all the caches must share a limited memory address space, and operating system memory mapping protocols may constrain available memory resources available to the caches, increasing the size of one cache may require reducing the size of another to avoid getting memory allocation failures.

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 multi-CPU 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 SHR_TARGET_FREE_FRAME_COUNT 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 number of 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.

Default: –1 (target is a percentage of total frames in cache; see above)
Minimum: –1
Maximum: 65536

STN_CHECKPOINT_INTERVAL

Internal parameter: **#StnCheckpointInterval**

STN_CHECKPOINT_INTERVAL sets the maximum interval (in seconds) between checkpoints. Checkpoints may be written more often, depending on other factors.

The internal parameter can be changed only by SystemUser.

Default: 300
Minimum: 5
Maximum: 1800

STN_DEAD_X_LOCKING_ENABLED

When set to False, STN_DEAD_X_LOCKING_ENABLED disables exclusive locking of dead objects that have not yet been reclaimed. This can be useful for applications that experience degraded commit performance when there is a large dead-not-reclaimed set.

CAUTION
To avoid the risk of corrupting your database, leave this parameter at its default value (TRUE) if your application ever uses object IDs to create committed structures.

Default: True

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 a user account exceeds these limits, the 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_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 203.

The internal parameter can be changed only by SystemUser.

Default: 1
Minimum: 0 (no threshold)
Maximum: 65536

STN_GC_SESSION_CONFIGURATION

STN_GC_SESSION_CONFIGURATION determines the GcGem configuration for the system when STN_GC_SESSION_ENABLED is set to True. The valid settings have these meanings:

1=Keep a single GcGem running.

2=Keep a Reclaim GcGem/Epoch GcGem pair running.

Default: 1

STN_GC_SESSION_ENABLED

Internal parameter: **#GcSessionEnabled**

If STN_GC_SESSION_ENABLED is set to True, then during startup, the Stone creates one or more Gem processes (GcGems) configured to reclaim unused space in disk pages and perform epoch garbage collection. The GcGem configuration is determined by the STN_GC_SESSION_CONFIGURATION configuration option.

The internal parameter can be changed only by SystemUser.

Default: True

Internal settings: 0 = False, 1 = True

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 forcibly aborts the Gem, sending it the error ABORT_ERR_LOST_OT_ROOT and then terminating it. (For more about LostOT behavior, see the following discussion of STN_GEM_LOSTOT_TIMEOUT.)

Negative timeouts are not allowed. Resolution of timeouts is one-half the specified timeout interval.

For more about how you might use STN_GEM_ABORT_TIMEOUT and STN_GEM_LOSTOT_TIMEOUT to handle non-responsive sessions, see "Commit Record Backlog Too Large" on page 357.

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 for a Gem running outside of a transaction to respond to an ABORT_ERR_LOST_OT_ROOT error before taking further action(s) to terminate the session.

If STN_GEM_LOSTOT_TIMEOUT is greater than or equal to zero, the Stone (more specifically, the Page Manager) poisons the session's shared page cache slot (and the slot used by the session's page server process, if any) and forcibly logs off the session. Thereafter, any attempt by the session or its page server to access the shared cache will raise a fatal error.

If STN_GEM_LOSTOT_TIMEOUT = -1, the Stone performs the above actions and also kills the session, by sending the session a SIGTERM signal. The SIGTERM signal is handled by the process and causes it to detach from the shared page cache and exit.

Negative timeouts other than –1 are not allowed. Resolution of timeouts is one-half the specified timeout interval.

GemStone never sends a SIGKILL (kill -9) signal to any process.

NOTE
Do not use SIGKILL to kill GemStone processes. Attempting to do so can result in repository crashes due to stuck spin locks.

For the reasons described here, you should avoid getting LostOT signals if at all possible, regardless of how you configure this parameter.

All actions described here are performed for any session, including sessions that use a remote shared page cache.

For more about how you might use STN_GEM_ABORT_TIMEOUT and STN_GEM_LOSTOT_TIMEOUT to handle non-responsive sessions, see "Commit Record Backlog Too Large" on page 357.

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 will cause the Stone to terminate the session. Negative timeouts are not allowed. Resolution of timeouts is one-half the specified timeout interval.

If STN_GEM_TIMEOUT is 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 one minute.

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 a number of login failures greater than or equal to STN_LOG_LOGIN_FAILURE_LIMIT 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_MAX_AIO_RATE

Internal parameters: **#StnMaxAioRate**

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_REMOTE_CACHES

The maximum number of remote shared page caches that the system may have. This limit includes both midlevel caches and "leaf" caches.

Default: 255
minimum: 0
maximum: 65535

STN_MAX_SESSIONS

STN_MAX_SESSIONS limits the number of simultaneous sessions (number of Gem logins to Stone). The actual value used by Stone is the value of this parameter or the number of sessions specified by the software license key file, whichever is less. This parameter is provided so that the number of users can 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 beyond 40, you may need to increase SHR_PAGE_CACHE_NUM_PROCS.

Recommended: 40, unless you are really using more sessions
Default: 40
Minimum: 1
Maximum: 8192

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 plus the number of extent replicates 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 four extents and two are replicated, setting STN_NUM_LOCAL_AIO_SERVERS to 3 causes each AIO page server to service (4 + 2) ÷ 3 = 2 extents.

NOTE
If STN_NUM_LOCAL_AIO_SERVERS *is greater than the number of extents plus the number of extent replicates, then by default, one AIO page server process is spawned for each extent or replicate.*

Under certain circumstances, multiple AIO page servers can help you achieve the maximum possible commit rate. A value greater than 1 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: 30

STN_NUM_SMC_QUEUES

Number of shared memory communication (SMC) queues used by gems and page servers to communicate with the stone. Valid values are: 0, 1, 2, 4 and 8.

A value of zero indicates the stone should compute the correct value based upon the maximum number of processes that can attach to the stone's shared page cache (SHR_PAGE_CACHE_NUM_PROCS) as follows:

SHR_PAGE_CACHE_NUM_PROCS	STN_NUM_SMC_QUEUES
< 256	1
256 - 1023	2
1024 - 2023	4
> 2024	8

Default: 0
Minimum: 0
Maximum: 8

STN_PAGE_MGR_COMPRESSION_ENABLED

Internal parameter: **#StnPageMgrCompressionEnabled**

Determines if the page manager will compress the list of pages it sends to remote shared page caches for removal. If set to TRUE, all lists of pages larger than 50 will be compressed before transmission. The same compressed list is used to send to all remote shared page caches; i.e., the compression operation is performed no more than once for each list of pages to be sent.

Has no effect on systems which do not use remote shared page caches.

Default: FALSE

STN_PAGE_MGR_PRINT_TIMEOUT_THRESHOLD

Internal parameter: **#StnPageMgrPrintTimeoutThreshold**

A threshold in real seconds used by the page manager to determine if a slow response from a remote shared page cache should be printed to the page manager log file. If a remote cache takes longer than this number of seconds to respond to the page manager, the page manager will print a message to the log file. If a remote cache takes less than this number of seconds to respond, no message is printed.

Note that this value controls the writing of log messages only. The connection to the remote cache will not be terminated by page manager unless STN_REMOTE_CACHE_PGSVR_TIMEOUT is exceeded.

Default: 5
Minimum: 0
Maximum: 3600

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, the Page Manager requests the pages from the Stone and processes them. Otherwise, the Page Manager waits 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: 16384

STN_PRIVATE_PAGE_CACHE_KB

STN_PRIVATE_PAGE_CACHE_KB sets the default size (in KB) of the Stone page cache.

Default: 1000
Minimum: 64
Maximum: 524288 (see note)

NOTE
The actual maximum for this parameter is dependent on the operating system memory mapping protocol and the sizes of other GemStone caches. See the note in SHR_PAGE_CACHE_SIZE_KB *for details.*

STN_RECOVERY_PAGE_RECLAIM_LIMIT

Internal parameter: **#StnRecoveryPageReclaimLimit**

STN_RECOVERY_PAGE_RECLAIM_LIMIT sets the maximum number of pages to reclaim when the Stone is playing a single transaction log record during system recovery or restoring from transaction logs. This value also determines the number of pages reclaimed during execution of `Repository>>restoreReclaimPages`. Decreasing this value postpones some page reclaiming, which can improve system recovery and restore performance, but at the expense of repository growth and heavier GcGem activity after the system is again operational.

Default: 2000
Minimum: 0
Maximum: 65536

STN_REMOTE_CACHE_PGSVR_TIMEOUT

STN_REMOTE_CACHE_PGSVR_TIMEOUT sets the time (in seconds) to wait for a response from a page server on a remote shared page cache. If no response is received within the timeout period, all Gems attached to that cache are logged off and a message is written to the Stone log.

Negative timeouts are not allowed. A timeout value of zero causes the Stone to wait forever.

Default: 15
Minimum: 0
Maximum: 3600

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.

Negative timeouts are not allowed. A timeout 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_REPL_TRAN_LOG_DIRECTORIES

STN_REPL_TRAN_LOG_DIRECTORIES lists the directories or raw disk partitions used for replicates of the transaction logs specified by STN_TRAN_LOG_DIRECTORIES. This list must either be empty, in which case logs are not replicated, or must have the same number of elements as STN_TRAN_LOG_DIRECTORIES. If the directory is on a remote host, the host name must be a network resource string. The size of the replicate log files is controlled by STN_TRAN_LOG_SIZES.

Default: Empty (logs are not replicated)

STN_REPL_TRAN_LOG_PREFIX

STN_REPL_TRAN_LOG_PREFIX sets file name prefix for transaction log file replicates, if any. A sequence number and `.dbf` suffix are added to the prefix; for example, the prefix "`repltranlog`" produces files named `repltranlog0.dbf`, `repltranlog1.dbf`,.... You can set this configuration option to permit multiple repository monitors to share a log directory without conflict.

Default: `repltranlog`

STN_SHR_TARGET_PERCENT_DIRTY

Internal parameter: **#StnMntShrPcTargetPercentDirty**

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 I/O rates.

Default: 33
Minimum: 5
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.

For further information, see Chapter 7, "Managing Transaction Logs."

Default: None. The system will not run unless a value is provided.
Initial setting: False

STN_TRAN_LOG_DEBUG_LEVEL

Internal parameter: **#StnTranLogDebugLevel**

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: 100 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 prefix for transaction log files. A sequence number and `.dbf` suffix are added to the prefix; for example, the prefix "`tranlog`" produces files named `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.

Default: Empty (the system will not run unless sizes are specified)
Minimum: 3
Maximum: 2147
Initial setting: 10, 10

A.4 Miscellaneous Internal Parameters

The internal parameters described in this section can by read by using the method `System class>>configurationAt:`. These parameters can be changed only by SystemUser.

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.

#StnTranLogOriginTime

`#StnTranLogOriginTime` is the time when the current transaction log was started.

Appendix B GemStone Utility Commands

The GemStone utility commands in this appendix are provided in the `$GEMSTONE/bin` directory. All but one (**waitstone**) can be executed with the **-h** option to display the usage information. For example:

```
    % copydbf -h

Usages: copydbf srcNRS dstNRS [-h|-l|-m|-P] [-C] [-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.

B.1 copydbf

copydbf	[**-l** | **-m** | **-P**] [**-C**] [**-f***filePrefix*] [**-n***netLdiName*] [**-p***pgsvrId*] [**-s***Mbytes*] [**-h**] *sourceNRS destinationNRS*
copydbf	**-i** [**-n***netLdiName*] [**-p***pgsvrId*] *sourceNRS*
copydbf	**-I** [**-n***netLdiName*] [**-p***pgsvrId*] *sourceNRS*
-l	Least-significant-byte ordering for the *destinationNRS*. This ordering is the native byte ordering for Intel x86 processors.
-m	Most-significant-byte ordering for the *destinationNRS*. This ordering is the default and is the native byte ordering for SPARC, IBM RS/6000, and HP PA-RISC machines.
-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.
-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`.
-f*filePrefix*	If *destinationNRS* is a file system directory, then *filePrefix* overrides the file name prefix that would be generated based on the contents of *sourceNRS*. If *destinationNRS* is other than a file system directory, this option has no effect.
-n*netLdiName*	The name of the GemStone network server; the default is "netldi66".
-p*pgsvrId*	The name of a specific `runpgsvr` (similar to `gemnetid`)
-s*MB*	The size to pre-allocate the destination file, in MBs. For instance, **-s**10 allocates at least 10 MBs to the created file. If the **-s** option is not specified, the output file is made as short as possible.
-h	Displays a usage line and exits.
-i	Information only. When this option is present without *destinationNRS,* information about *sourceNRS* is printed

	without performing a file copy. If both **-i** and *destinationNRS* are present, this option is ignored and a copy is performed.
-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.
sourceNRS	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.
destinationNRS	The destination file, directory, or raw partition as a GemStone network resource string. If the destination is a file system directory (the trailing / is optional), a file name is generated and appended to *destinationNRS* based on the type and internal fileId of the source. Use of `/dev/null` as the destination is supported only for files as a means of verifying that the file is readable. Use of a tape device as the destination is not supported.

The **copydbf** utility requires exclusive write 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 that change byte ordering, or copies to and from raw disk partitions. GemStone repository files can be written in any byte ordering, but non-native byte ordering will make GemStone run more slowly. 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 file name 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`. The prefix can be changed through the **-f** option.

A message describing the source and destination files is printed to standard error prior to 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 .

[08/05/11 19:15:41.881 PDT]
Source file: /users/GemStone6.6/data/extent0.dbf
   file type: extent  fileId: 0
   Last checkpoint written at: 08/05/11 15:43:52 PDT.
Destination file:  ./extent0.dbf
   Clean shutdown, no tranlog needed for recovery,
   last tranlog written to had fileId 21 ( tranlog21.dbf ).
   File size is 17.8 MBytes (2176 records).
```

The same source file information (but not the size) can be obtained without making a copy by using the second form of the command, **copydbf -i** *sourceNRS*. In this usage, *destinationNRS* must be omitted. The information for an extent also shows the oldest transaction log that would be need to recover from a system crash, and for a backup, to restore subsequent transaction. The first example is applied to an extent, and the second, to a backup:

```
c:\> copydbf -i extent0.dbf

[08/15/11 11:09:19.349 PDT]
   Source file: extent0.dbf
  file type: extent  fileId: 0
    Last checkpoint written at: 08/15/11 11:07:54 PDT.
    Oldest tranlog needed for recovery is fileId 5 (
tranlog5.dbf ).
```

```
c:\> copydbf -i back4.dat

[08/15/11 11:05:05.159 PDT]
   Source file: back4.dat
  file type: backup  fileId: 0
     The previous file last recordId is  -1.
   Destination file: /dev/null
     Full backup started from checkpoint at: 08/15/11
09:04:49 PDT.
  Oldest tranlog needed for restore is fileId 5 (
tranlog5.dbf ).
     Backup was created by GemStone Version: 6.6.0 .
```

To obtain the size of a repository file in a raw partition, use **copydbf** *sourceNRS destinationNRS*.

A listing of all checkpoints recorded in a transaction log can be obtained by using **copydbf -I** *sourceNRS*. This information is helpful in restoring a GemStone backup to a particular point in time. For example,

```
c:\> copydbf -I tranlog6.dbf

[08/05/11 19:17:31.178 PDT]
Source file: tranlog6.dbf
  file type: tranlog  fileId: 6
  The file was created at:  08/05/11 10:56:25 PDT.
  The previous file last recordId is  65.
  Scanning file to find last checkpoint...
Destination file:  /dev/null
  Checkpoint 1 started at: 08/05/11 10:56:29 PDT.
   oldest transaction references fileId 5 ( tranlog5.dbf ).
  Checkpoint 2 started at: 08/05/11 10:56:37 PDT.
   oldest transaction references fileId 5 ( tranlog5.dbf ).
  Checkpoint 3 started at: 08/05/11 11:07:49 PDT.
   oldest transaction references fileId 5 ( tranlog5.dbf ).
  Checkpoint 4 started at: 08/05/11 11:21:20 PDT.
   oldest transaction references fileId 5 ( tranlog5.dbf ).
  File size is 10240 bytes (20 records).
```

You can pre-allocate disk space in the destination file by using the **-s** option. For instance, **-s10** would allocate at least 10 megabytes 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 "C:/path/extent0.dbf_copy," via the standard GemStone network server "netldi66" using TCP protocol:

Bourne shell:

```
$ copydbf $GEMSTONE/data/extent0.dbf \
!tcp@node#auth:username@password#dbf!/users/extent0.dbf_copy
```

or C shell:

```
% 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.) First, you make a local copy of the distribution extent. Using **copydbf** requires that you have write permission on the extent (or transaction log) that you are copying.

```
% cp $GEMSTONE/bin/extent0.dbf tempDirectory
% chmod +w tempDirectory/extent0.dbf
% copydbf tempDirectory/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/
```

B.2 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 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
-u *user*	only list servers started by *user*
-t *secs*	wait *secs* seconds for server to respond (only with **-v**); default is 2 seconds
-m *host*	only list servers on machine *host*; default is '.' which is the local host
[**-n**] *name*	only list the server *name*

This 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
		killedserver process does not exist
Version	GemStone version of the server	
Owner	account name of user who created server	
Started	date and time server was started	
Type	one of Netldi, Stone, or cache (shared page cache monitor)	

Name	server's name

The following additional columns are print by **-l** or **-x**:

Pid	process id of server's main process
Port	port number of server's listening socket

The **-x** prints the preceding attributes on separate lines and adds lines for the following as appropriate:

options	options used when server was started
logfile	full path of server's log file if it exists
sysconf	Stone's system configuration file
execonf	Stone's executable configuration file
GEMSTONE	root of the product tree used by the server

Multiple hosts can be specified by using **-m** *host* more than one time. 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, ordinarily **gslist** must 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.

Multiple server names can be specified by using **-n** *name* more than one time. 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 *timeout*.

B.3 pageaudit

pageaudit	[*gemStoneName*] [**-e***exeConfig*] [**-z***systemConfig*] [**-f**] [**-h**]
gemStoneName	name of the GemStone repository monitor; the default is "gemserver66-audit". Network resource syntax is not permitted.
-d	disable audit of data pages
-e*exeConfig*	the GemStone executable-dependent configuration file
-z*systemConfig*	the GemStone system configuration file
-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*, **-e***exeConfig*, and **-z***systemConfig* determine which configuration files **pageaudit** reads; see "How GemStone Uses Configuration Files" on page 394 for more information. The arguments 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 the **-f** switch is specified, **pageaudit** prints all errors possible. Without **-f**, the default is to stop after the first error is found.

pageaudit audits both data and non-data pages. To disable audit of page pages, use the -d option.

This utility can take a long time to run, so it is best to run it as a background job.

pageaudit creates a temporary extent in the directory specified by DBF_SCRATCH_DIR. This requires that disk space is available in this directory; allow 2% to -5% of the repository extent sizes.

For additional information about **pageaudit** and a description of its output, see "To Perform a Page Audit" on page 227.

B.4 removedbf

removedbf	*dbfNRS* [-**n** *netLdiName*] [-**p** *pgsvrNetId*] [-**h**]
dbfNRS	the GemStone repository file name or device, as a network resource string, for the repository to be removed
-***n*** *netLdiName*	the name of the GemStone network server, default is "netldi66"
-**p** *pgsvrNetId*	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 give an NRS (network resource string) for the file to be removed. (A local machine filespec is a subset of an NRS).

If a file in the file system is specified, this command is equivalent to the **rm** command. If a raw disk partition is specified, GemStone meta data 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 193.

The options for this command generally are not needed for a standard GemStone configuration.

B.5 startcachewarmer

startcachewarmer	[**-h**] [**-n***numGems*] [**-p***password*] [**-s***stone*] [**-u***userID*] [**-w***delayTime*] [**-W**]
-h	Displays a usage line and exits.
-n*numGems*	Number of Gem sessions to start (default: 1).
-p*password*	GemStone password for logging in Gems (default: 'swordfish').
-s*stone*	Name of the running Stone (default: gemserver66).
-u*userID*	GemStone user for logging in Gems (default: 'DataCurator').
-w*delayTime*	Wait *delayTime* 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 the object table 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.

Each cache warming Gem reads a portion of the object table into the shared page cache. Cache warming Gems automatically terminate when the shared cache becomes full or when their portion of the object table has been completely loaded, whichever occurs first.

The cache warmer runs outside of a transaction and responds to SigAborts. Cache warmers are safe for use during production.

B.6 startnetldi

startnetldi	[*netLdiName*] [**-l***logFile*] [**-t***timeout*] [-**a***name*] [-**p***low***:***high*] [**-n**] [-**g** | **-s**] [-**d**] [-**h**]
netLdiName	the name of the GemStone network server. It may contain digits but it must not be entirely numeric. Network resource syntax is not permitted. The default is "netldi66".
-l*logFile*	the logged output of the NetLDI; the default is `/opt/gemstone/log/`*NetLdiName*`.log`
-t*timeout*	seconds to wait for a spawned client to start, such as a Gem; default is 30 seconds.
-a*name*	captive account; all child processes created by the netldi will belong to the account named *name*. By default, child processes belong to the client's account.
-**p***low***:***high*	the low and high ports in the pool of ports to use in creating processes.
-n	no *ad hoc* processes shall be created (ad hoc processes are ones not listed in `$GEMSTONE/sys/services.dat`).
-**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; authentication required for ALL `netldi` accesses
-d	debug mode; inserts more extensive messages in the log file
-**h**	displays a usage line and exits

Start a GemStone network server *netLdiName* with a time-out given in seconds. This server spawns GemStone processes in response to login requests from remote applications and requests for remote repository access from Stone repository monitor processes. The name and time-out defaults may be changed via optional command line arguments. On slow or heavily loaded machines, use a larger time-out value, typically 300 seconds. By default, this server writes its log information to `/opt/gemstone/log/`*netLdiName*`.log`.

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):

```
% 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 102.

For assistance with start-up failures, refer to “To Troubleshoot NetLDI Startup Failures” on page 134.

B.7 startstone

startstone	[*gemStoneName*] [**-l***logFile*] [**-e***exeConfig*] [**-z***systemConfig*] [**-h**] [**-N**] **[-R]**
gemStoneName	name of the GemStone repository monitor, default is "gemserver66". Network resource syntax is not permitted.
-l*logFile*	the logged output of the stone; the default is (1) a `GEMSTONE_LOG` environment variable, if defined (2) `$GEMSTONE/data/`*gemStoneName*`.log`
-e*exeConfig*	the GemStone executable-dependent configuration file
-z*systemConfig*	the GemStone system configuration file
-h	displays a usage line and exits
-**N**	Start up when no transaction logs are available. This option should be used only for disaster recovery.
-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.

Open the GemStone repository specified by the relevant configuration files. The three arguments *gemStoneName*, **-e***exeConfig*, and **-z***systemConfig* determine which configuration files **startstone** reads; see for more information. Arguments to this command are optional and are not needed for a standard GemStone configuration.

The -**N** option is intended for recovery from a disaster that has corrupted or destroyed the transaction log files. If the transaction logs specified in the configuration file cannot be found and if the repository needs recovery, 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. This option may be used 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 start up.

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. Topaz must then be invoked to restore from transaction logs (if available) and commit the restored state. If the extents against which stone is being started require recovery, then use of the **-R** option will result in an error and repository monitor will not start. This error may be overridden by using both **-N** and **-R**.

For assistance with start-up failures, refer to “To Troubleshoot Stone Startup Failures” on page 127.

B.8 stopnetldi

stopnetldi	[*netLdiName*] [**-h**]
netLdiName	the name of the GemStone network server; the default is "netldi66". 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.

B.9 stopstone

stopstone	[*gemStoneName* [*gemStoneUserName* [*gemStonePassword*]]] [**-i**] [**-h**]
gemStoneName	the name of the GemStone repository monitor, by default "gemserver66". Network resource syntax is not permitted.
gemStoneUserName	a privileged GemStone user account name, such as "DataCurator" or "SystemUser"
gemStonePassword	the GemStone user account password.
-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 the immediate (**-i**) option is specified, the repository monitor is shut down even if there are GemStone users logged in. If the *gemStoneName*, *gemStoneUserName*, and *gemStonePassword* are not supplied 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, or if you are at the limit of the number of licenced or configured logins. For assistance, refer to "To Troubleshoot Session Login Failures" on page 139.

B.10 topaz

topaz [-r]	[**-i**] [**-n***netLdiName*] [**-h**]
topaz -l	[**-i**] [**-n***netLdiName*] [**-e***exeConfig*] [**-z***systemConfig*] [**-h**]
-l	invoke the linked version of Topaz
-r	invoke the RPC (remote procedure call) version of Topaz
-i	ignore the initialization file, `.topazini`
-n *netLdiName*	the name of the GemStone network server; the default is (1) a `GEMSTONE_NRS_ALL` environment variable (2) "netldi66"
-e*exeConfig*	the GemStone executable dependent configuration file (applies only to linked sessions)
-z*systemConfig*	the GemStone system configuration file (applies only 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 version of Topaz. The arguments **-e***exeConfig*, and **-z***systemConfig* determine which configuration files **topaz -l** reads; see "GemStone Configuration Options" on page 393 for more information. 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*.

B.11 waitstone

waitstone	[*gemStoneName* | *netLdiName*] [*timeout*]
gemStoneName	the name of the GemStone repository monitor, by default `gemserver66`
netLdiName	the name of the GemStone network server, by default `netldi66`
timeout	how many minutes to wait for GemStone to initialize before reporting a problem. Default is 0 minutes. 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. The *gemStoneName* and *netLdiName* arguments may be specified 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 on-line manual page.

Appendix C 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.

C.1 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 "gemserver66" running on node "handel", you would specify two NRS strings:

```
topaz> set gemstone !@handel!gemserver66
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. For example, under the UNIX C shell, you must escape an exclamation point (!) with a preceding backslash (\) character:

```
% waitstone \!tcp@oboe\!gemserver66 -1
```

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!gemserver66
% waitstone "!tcp@oboe#auth user@password!gemserver66"
```

C.2 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 454. 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.

C.3 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**)

C.4 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 `netldi66`. When you specify the **#netldi** option, the *nrs-identifier* (page 454) 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`" (defined below) is used. (See the earlier discussion of *nrs-identifier*.)

#log sets the name of the log file of the network resource. It has no effect if the resource already exists. If the log name is a relative path, it is relative to the working directory. If a log name is not set, the pattern "`%N%P%M.log`" (defined below) is used. (See the earlier discussion of *nrs-identifier*.)

The argument to **#dir** or **#log** can contain patterns that are expanded in the context of the created resource. The following patterns are supported:
%Hhome directory

%Mmachine's network node name
%Nexecutable's base name
%Pprocess pid
%Uuser 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/sys/services.dat`, and then in the user's home directory for an executable having that name. The NetLDI returns the network address of the service. (Authorization is needed to create a new process unless the NetLDI is in guest mode.) The **#task** resource modifier is also used internally to create page servers.

#dbf is used to access a database file. The *nrs-body* is the file spec of a GemStone database file. The NetLDI creates a page server on the given node to access the database and returns the network address of the page server. (Authorization is needed unless the NetLDI is in guest mode).

#spawn is used internally to start the garbage-collection Gem process.

#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.

Appendix

D GemStone Kernel Objects

This appendix describes the predefined objects that are located in a freshly installed GemStone repository.

D.1 Users

The AllUsers object, an instance of UserProfileSet, contains the UserProfile objects of the users that are known to the repository. Initially, it has four elements: **SystemUser**, **DataCurator**, **GcUser**, 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 the reclaim and epoch garbage collection.

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.

D.2 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 three keys: #UserGlobals (corresponding to the dictionary itself); #MinutesFromGmt (different from **MinutesFromGMT** in the Globals dictionary and not used in this release); 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 465) 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. Objects in this dictionary can be read by one group (#Subscribers) and modified by another group (#Publishers). The #Publisher group can also be a subset of the #Subscriber group. The Published dictionary is primarily an example, so it provides minimal object deployment capability. The Published dictionary is stored in the Published Segment.

D.3 Non-Numeric Constants

true. This object (an instance of Boolean) is defined in the Globals dictionary, and is stored in the System Segment.

false. This object (an instance of Boolean) is defined in the Globals dictionary, and is stored in the System Segment.

nil. This object (an instance of UndefinedObject) is defined in the Globals dictionary and is stored in the System Segment.

D.4 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

D.5 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 six 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. No groups are initially authorized to read or write in this Segment.

Objects in the DataCuratorSegment include the Globals dictionary, the SystemRepository object, all 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.

NOTE
When GemStone is installed, only the DataCurator is authorized to write in this Segment. To protect the objects in the DataCurator Segment

against unauthorized modification, other users should not write in this Segment.

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 residing in the DataCurator Segment that contains information regarding conversions and updates applied to the repository.

MinutesFromGMT. This SmallInteger is defined in the Globals dictionary and is reserved for use in future releases.

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.)

D.6 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 who have been authorized to read or write in one or more Segments. When GemStone is first installed, AllGroups contains the single symbol #System.

AllSymbols. This CanonicalStringDictionary is defined in the Globals dictionary, and is stored in the DataCurator Segment. Initially, it contains references to all Symbols created with GemStone itself. AllSymbols is in the DataCurator segment and is readable by all users.

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 three UserProfiles: SystemUser, DataCurator, and GcUser.

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 seven 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”, also used for AllSymbols, 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 can not create instances. Some of these classes (like System) have no instances at all. A fresh GemStone installation already contains all possible instances of others (like Boolean and UndefinedObject).

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
	#AllSymbols	CanonicalStringDictionary
	#AllUsers	UserProfileSet
	#ConfigurationParameterDict	SymbolKeyValueDictionary
	#ErrorSymbols	SymbolDictionary
	#GemStoneError	SymbolDictionary
	#Globals	SymbolDictionary
	#InstancesDisallowed	IdentitySet
GemStone Internal Objects	#AsciiCollatingTable	ByteArray
	#ConversionDict	SymbolDictionary
	#ConversionReservedOopMap	Array
	#DbConversionStatus	Array
	#DoubleByteAsciiCollatingTable	DoubleByteString
	#GcCandidates	RcQueue
	#GcCandidatesCount	RcCounter
	#GcHints	UndefinedObject
	#GsIndexingSegment	Segment
	#GcWeakReferences	Array
	#ImageVersion	SymbolDictionary
	#OldAllUsers	AbstractUserProfileSet
	#_remoteNil	UndefinedObject
	#SecurityDataSegment	Segment
	#SharedDependencyLists	DepListTable
	#VersionParameterDict	SymbolKeyValueDictionary
plus all kernel classes		

D.7 Current TimeZone

Each instance of DateTime includes a reference to a TimeZone object, which handles the conversion from the internally stored Greenwich Mean Time (GMT, also referred to as UTC or Coordinated Universal Time) and the local time. TimeZones encapsulate the daylight savings time (DST) rules, so a given GMT time is adjusted to local time based on TimeZone and the specific date. TimeZones are also used to calculate the internal stored GMT for newly created DateTime instances.

Each session has a current TimeZone, which is, by default, the system-wide default TimeZone. The current TimeZone is used to display times, and in DateTime creation when using methods that do not explicitly specify the TimeZone. In most cases, the initial system-wide default should be replaced as part of application installation or configuration. For details, see the *GemStone/S Installation Guide*.

GemStone uses the public domain **zoneinfo** database to create TimeZone, loading the information from platform- and language-independent binary files. If the rules change for the TimeZone that your application uses, you should obtain new binary files and use them to update the GemStone internal TimeZone instances. The process for obtaining new binary files is described in the following section, "Zoneinfo".

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 Solaris and Linux, you can create the TimeZone instance based on the current machine configuration, as follows:

  ```
  newTZ := TimeZone fromOS
  ```

- **GemStone's time zone database**. Using the interactive script `tzselect` (page 469), 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 instance 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.

For more information about ZoneInfo, see the website:

```
http://www.twinsun.com/tz/tz-link.htm
```

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.

The public domain tz database is distributed as text files, which can be compiled into binary files using the associated tz compilers. GemStone's TimeZone implementation uses the compiled binary form, which is also used by the Solaris and Linux operating systems.

If you need updated tz files, as in the case where time zone rules change unexpectedly, you may update your tz database by downloading the most recent source files, and compiling them into binary form using the zic timezone compiler. GemStone provides the zic compiler as a convenience. For details, see "zic" on page 469.

The most recent versions of the tz data files can be downloaded from:

```
ftp://elsie.nci.nih.gov/pub/tzdata*.tar.gz.
```

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.

NOTE
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 can download the source code for these utilities from the following site:

```
ftp://elsie.nci.nih.gov/pub/tzcode*.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

The tzselect utility allows to you 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, see the tzselect man page.

zdump

```
zdump [-v] [-c cutoffyear] [zonename...]
```

The zdump utility prints time zone information. It prints the current time for each time zone (*zonename*) listed on the command line.

Specifying an invalid *zonename* 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 displays the entire contents of the time zone database for the given *zonename*.

For further details, including command line options, see the zdump 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, including command line options and the structure of the source code files, see the zic man page.

Appendix

E

Environment Variables

This appendix lists the environment variables used by GemStone. The list has two parts: variables intended for public use, and variables that are reserved for internal use.

E.1 Public Environment Variables

The following environment variables are intended for user 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/GemStone6.6-sparc.Solaris`.
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 397.

GEMSTONE_LANG
: The name of a translated message file in `$GEMSTONE`/bin. (This file is not provided with GemStone.) For further information, see “Specifying a Language” on page 476.

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 222.

GEMSTONE_MAX_FD
: Limits the number of file descriptors requested by a GemStone process. For further information, see “Estimating File Descriptor Needs” on page 38.

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 107.

GEMSTONE_SYS_CONF
: Location of a system-wide configuration file; see “How GemStone Uses Configuration Files” on page 394.

GS_AutoMountPrefix
: The string that a UNIX automounter prefixes on a file path; the default prefix if this variable is not set is `/tmp_mnt`.

GS_CFTIME
: The display format string for use when displaying a date and time in a log header or entry. If not set, the current envionrments Locale date and time display will be used. If locale is not set, US default is used.GS_CFTIME is set to a C format variable display string, such as `'%Y-%m-%d %H:%M:%S'`.

GS_CORE_TIME_OUT
: 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, usually derived from the core file. Under some circumstances, however, core files may be impractically large or otherwise unusable; in such cases, a stack trace can be extracted directly from the failing but not yet terminated process by attaching a debugger to it. Increasing the value of this variable increases 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_DISABLE_KEEPALIVE

A non-empty string disables the network keepalive facility. For further information about keepalive, see “Disrupted Communications” on page 101.

GS_DISABLE_SIGNAL_HANDLERS

To enable this variable, set it to any value; the actual setting does not matter. When enabled in the environment for a Gem process, the Gem sessions will not attempt to handle SIGSEGV, SIGBUS, or SIGILL signals, but will shut down immediately. This avoids side effects with user action or client Smalltalk code.

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`.

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 exists 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 `System class>>performOnServer`.

E.2 Reserved Environment Variables

The following environment variables are reserved for internal use. Customers should not define these variable 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

netldi*nn*

runpgsvr

Appendix

F

Localization

This appendix addresses the following topics related to localization:

- Translation files for messages (page 476)

  How to create and compile the source of the GemStone error file and how to create a similar file for another language.

- GemStone Smalltalk class Locale (page 485)

  How to obtain and override operating system locale information in GemStone.

- Extended Character Set support (page 486)

  How to process and collate String data that includes extended Characters (that is, beyond the basic set of 256 Characters).

F.1 Translation Files for Messages

GemStone prints error and advisory messages by inserting arguments into text that it extracts from a language-dependent, pre-compiled error file. GemStone currently provides only one such file, `$GEMSTONE/sys/english66.err`. This section explains how to create and compile the source of that file and how you can create a similar file for another language.

The message system described in this appendix operates outside of object memory (that is, when the user is not logged in to a GemStone session). When users are logged in, messages can be translated through GemStone Smalltalk's LanguageDictionary. For further information, see the *GemStone Programming Guide*.

Specifying a Language

All GemStone executables infer the user's native language from the environment variable GEMSTONE_LANG, which should point to a file in $GEMSTONE/`sys` that has the name *language*`66.err`. If GEMSTONE_LANG is not defined, the default value $GEMSTONE/`sys/english66.err` is used.

The Message Compiler

GemStone messages are stored in a pre-compiled error file, `$GEMSTONE/sys/`*language*`66.err`, for efficient retrieval at run time. Users who want to translate the default `english66.err` into another language can regenerate the language source by running the **msgcom** utility to produce a text file. After translation, you can compile the new language source by again invoking **msgcom**. The syntax is

`msgcom totext` *infile outfile*
`msgcom fromtext` *infile outfile* [*hashsize*]

where *hashsize* is an optional parameter for use with **fromtext** to override the default hashtable size in the resulting error file. For example, the following re-creates the source for the default error file and places it in the file `english.lang`:

Example F.1

```
% $GEMSTONE/sys/msgcom totext $GEMSTONE/sys/english66.err \
english.lang
...
            wrote 187044 characters
```

The Language Source File

This section describes the language source file syntax, semantics, and rules for resolving conflicts in argument cardinalities. The compiled contents of a language file are read at run time to create a language-specific error message.

Example F.2 contains three parts that show the steps by which the **startstone** utility displays a pair of start-up messages familiar to GemStone administrators:

- The calls to $GEMSTONE/`bin/saymessage`, which specify an error symbol in the language file and following it with arguments (the paths or the server name).
- The language file entries, which map the arguments to message text.
- The resulting messages for sample arguments.

Example F.2

```
saymessage startstone.info $GEMSTONE_SYS_CONF $GEMSTONE_EXE_CONF
saymessage startstone.already "$stoneName"

startstone.info (1 2):\
startstone[Info]:\n\
    GEMSTONE_SYS_CONF=%s\n\
    GEMSTONE_EXE_CONF=%s\n

startstone.already (1):\
startstone[Info]: checking if server %s is already running...\n
```

```
startstone[Info]:
     GEMSTONE_SYS_CONF=/user2/Gemstone6.6/data/system.conf
     GEMSTONE_EXE_CONF=/user2/application.conf
startstone[Info]: checking if server gemserver66 is already running...
```

Language File Syntax

A language file provides a mapping from an error symbol to a piece of text and a description of the ordering of arguments. This is reminiscent of the Smalltalk mechanism with LanguageDictionary and draws upon the `printf()` syntax.

A language file is translated as follows:

1. If the last character on a line is backslash (\), the next line (if any) is appended to it. This process is repeated if necessary. The resulting line must be less than 4096 characters.
2. If the line is empty, or the first non-blank character on the line is a sharp (#), the line is discarded.
3. Otherwise, the line is a mapping line, and has the following syntax:

   <*mapping_line*> ::= <*symbol*> [<*ordering*>] "**:**" <*result_text*>

   <*ordering*> ::= "(" { <*numeral*> } ")"

   where a <*symbol*> is any run of non-blank characters excluding "(" or ":", <*result_text*> is arbitrary text (translation rules given below), and <*numeral*> is zero or more non-negative base-10 numerals that determine the order in which arguments are substituted in <*result_text*>. The use of <*ordering*> is described further just ahead.

Language File Semantics

A C program creates an error message by passing an error symbol and a list of arguments. The number and types of the arguments are known. The order in which the arguments are passed is known as the "canonical ordering." In Example F.2 (on page 477), the first error symbol is "`startstone.info`" and there are two arguments, which are the configuration file paths.

Example F.2 also illustrates the GemStone practice of creating the error symbol from a combination of a module or executable name ("startstone") and a subordinate identifier (such as "checking"). This convention helps to organize the large number of messages.

If the error symbol cannot be found, a "desperation message" is printed. For example, if the error symbol is "frobnitz", and the arguments are 1, 2, and "foo", the resulting desperation message is:

```
[frobnitz 1 2 foo]
```

This desperation message has two important features:

- The error symbol itself and all of its arguments are displayed.
- Because GemStone error messages are often composed out of other error messages, the surrounding brackets indicate the scope of the message and its arguments. The brackets also provides an indication that the message is not translated.

The ordering clause in a mapping line specifies the order in which the actual arguments are to be substituted into the result text. A "1" in the ordering clause refers to the first argument in the canonical ordering, a "2" to the second, etc.

The mapping process effectively gives three different argument cardinalities:

ACTUAL	The number of arguments actually passed.
EXPECTED	The number of arguments expected, implied by the largest integer listed in the ordering clause.
USED	The number of arguments actually used in the result line.

The following mapping line uses three arguments but switches their order from that in the calling statement:

```
frobnitz (2 1 3): The %d is not %s in the %s.\n
```

A matching call to the `saymessage` executable would contain three actual arguments:

```
saymessage frobnitz sleeping 29 bed
```

and the resulting text would be this:

```
The 29 is not sleeping in the bed.
```

If a message has more than GCI_MAX_ERR_ARGS arguments (defined in `$GEMSTONE/include/gci.ht`), an error results. Numerals that are higher than the actual number of arguments are permitted, but will print as "0" in the resulting message.

Arguments can be omitted in an ordering clause or used more than once. However, the total number of arguments listed in the ordering clause is still limited to GCI_MAX_ERR_ARGS.

Conflicting Argument Cardinalities

If the ACTUAL, EXPECTED, and USED argument cardinalities differ, the resulting error message attempts to do something reasonable. In the following discussion, the calling statement has the three arguments from the previous example (unless stated otherwise):

```
saymessage frobnitz sleeping 29 bed
```

1. ACTUAL == EXPECTED < USED

   Suppose the language file uses four arguments:

   ```
   frobnitz (2 1 3):The %d is not %s %s in the %s.\n
   ```

   Then the resulting message will be

   ```
   The 29 is not sleeping bed in the %s.
   ```

   Argument patterns with no actual argument are simply not translated.

2. ACTUAL == EXPECTED > USED

   Suppose the language file uses only two arguments:

   ```
   frobnitz (2 1 3):The %d is not %s.\n
   ```

   Then the resulting message will be

   ```
   The 29 is not sleeping.
   ```

   The format of the messages acknowledges the presence of the third argument and intentionally fails to print it. This is acceptable.

3. ACTUAL > EXPECTED

   Suppose the language file contains the entry:

   ```
   frobnitz (2 1):The %d is not %s.\n
   ```

   but the symbol “frobnitz” is invoked with actual arguments “sleeping”, “29”, “bed”, and “foo”. Then the result message will be

   ```
   ["The %d is not %s.\n" frobnitz sleeping 29 bed foo]
   ```

   The system knows that more arguments are present than would actually be printed, so the “desperation” message is printed. Notice that the result text is included at the beginning of the message.

4. ACTUAL < EXPECTED == USED

Suppose the language file contains the entry:

```
frobnitz (2 1 3):The %d is not %s in the %s.\n
```

but the symbol "frobnitz" is invoked with only two arguments, "sleeping" and "29". Then the result message will be

```
The 29 is not sleeping in the [null].
```

The third argument is known to be bogus and is printed as a "[null]".

5. ACTUAL == USED > EXPECTED

   Suppose the language file using three arguments but only has ordering for (expects) two:

   ```
   frobnitz (2 1):The %d is not %s in the %s.\n
   ```

   If the symbol "frobnitz" is invoked with actual arguments "sleeping", "29", and "bed", the result is

   ```
   ["The %d is not %s in the %s.\n" frobnitz sleeping 29
   bed]
   ```

   This example is analogous to case 3. The language file entry declares that it will only print two arguments; hence, it is considered better to print the desperation message than to omit the third argument. As in case 3, the result text also is printed in the desperation message.

6. ACTUAL == USED < EXPECTED

   Suppose the language file contains an entry that uses three arguments but has ordering for (expects) four:

   ```
   frobnitz (2 1 3 4):The %d is not %s in the %s.\n
   ```

   If the symbol "frobnitz" is invoked with actual arguments "sleeping", "29", and "bed", then the resulting message is

   ```
   The 29 is not sleeping in the bed.
   ```

   The system knows the fourth argument is bogus, but nevertheless it uses the language file entry to format the message.

The Result Text

The result text is translated according to `printf()` rules. To recapitulate the printf() rules:

"\" escapes the character that follows it, with these exceptions:
\b = backspace.

\f = form feed,
\n = newline,
\r = carriage return,
\t = tab,
\v = vertical tab (control-K, ASCII 11)
\nnn = ASCII character with octal value nnn

"%"	indicates the beginning of an argument field. "%%" refers to a literal "%" in the result text. Otherwise, an input argument is printed in place of the argument field.

An argument field can be quite complex; for example, "%#-32lX".

In translating arguments, the following exceptions to the `printf()` rules apply:

- If more argument fields are specified in the result text than are actually passed, the superfluous argument fields are not replaced.
- If the type specified in an argument field varies egregiously from the actual type passed (such as string vs. int vs. float), the actual argument is printed with a minimal argument field that agrees with its actual type (such as %s, %ld, or %g).

Language File Errors

It is not reasonable to try to diagnose language file problems in a language-independent manner. Thus, problems in the language file are always diagnosed in English and printed on standard error. This section provides several examples along with brief explanations:

`ErrMsgInit (warning): no language file %s`

The given language file cannot be opened.

`symbol missing in %s near line %d`

The entire line was scanned without finding a colon (:).

`")" missing in %s near line %d`

An ordering term was found (left parenthesis), but no matching right parenthesis was found.

`")" expected in %s near line %d`

Something besides integers was found in an ordering term.

`message missing in %s near line %d`

The entire line was scanned without finding a colon (:).

`numeral out of range in %s near line %d`

An ordering numeral with value greater than GCI_ERR_MAX_ARGS was specified.

`ErrMsg!findErrSym: %s defined again near line %ld\n”,`

An error symbol is multiply defined. The point of the second definition is given.

`too many arguments in %s near line %d`

The list of arguments is too long.

Creating New Message Files

This section discusses the subtleties, pitfalls, and stylistic points in writing error messages.

Most importantly, **do not** translate the symbol itself. This string is the key used by the message subsystem to locate the message text.

It is probably easiest to start by using **msgcom** to create a text version of the existing messages the basis for a new error message file. See “The Message Compiler” on page 476.

If a GemStone message has arguments, they have been ordered in the executables and shell scripts in an ordering convenient for English grammar. If you need to reorder arguments (for instance, adjective after noun in Romance languages), modify the ordering clause.

Note very carefully the presence or absence of newlines (\n) at the end of messages. Some messages are intended to be used within other messages, and thus do not end with a newline.

Please be very careful to use grammatically complete and correct language constructions. It is not possible for GemStone Engineering to proofread and correct your work. It has been the goal of GemStone Engineering to use complete English sentences where possible.

Formatting Tips

Quite a bit of thought has gone into formatting of some of the messages, especially those in two categories:

- Messages written to Stone's standard output (GemStone log messages) are uniformly indented three spaces, with other explanatory data indented beyond that. When you translate these messages, keep in mind what the aggregate of these messages in the log file will look like.
- HostFaultHandler messages (core dump, segmentation violation, child process death, and so forth) are all formatted to appear within a "box." First of all, they do not have trailing newlines, since the box drawing code needs to right-pad the text. Second, the lines typically have a "title" and "contents" parts, and these are all formatted to line up in an aesthetically pleasing manner.

Shell Level Access

The language files can be accessed from the shell command line via a GemStone executable, $GEMSTONE/`bin/saymessage`. This executable is used by GemStone shell scripts to print their messages in a language-independent fashion. It is also handy for checking the result of a particular message symbol during the translation process.

The arguments to `saymessage` are a message symbol followed by message arguments (if any) in canonical order. The arguments are passed as strings.

Untranslated Messages

A few categories of messages in the system don't translate to other languages:

- GemStone Copyright Notice—This copyright notice has its text embedded in the executable for legal purposes. Furthermore, the copyright notice is binding in all countries, even though it is in English. Thus, it is not possible to affect (or suppress) the copyright message.
- Assertion Failures—There are places, even in the production executables that we ship, where if things get out of hand we will crash the executable with a (typically) cryptic message and a core file. These failures always indicate a problem that should be referred to GemStone Customer Support. These messages do not go through the error message subsystem and consequently cannot be translated.
- Bootstrap and Error Subsystem Messages—The shell scripts try to figure out the current language and to use that language in their messages. However, if the installation is defective (for instance, no `dirname`(1) command, or no $GEMSTONE/`bin/saymessage` executable), we have no recourse but to print a message in English. By the same token, the installation script has a window during which its messages must appear in English. Similarly, if there

are errors in the language file (syntax errors, error symbol defined more than once, and so forth), we print a message in English.

Any other messages that don't translate are oversights on the part of GemStone Engineering and should be reported as bugs.

Message Context

It may not be clear from context what the purpose of many of the error messages is. Please direct inquiries to GemStone Customer Support. We will provide you with an explanation, and will insert explanatory comments into the next release of our English language version of the text.

Note that, by the same token, it is important that the comments in the language file also be translated, where possible. This is to facilitate the potential for second-generation language files (that is, English to French to Polish to Finnish to....).

F.2 Class Locale

The class Locale allows you to obtain operating system locale information and use or override it in GemStone. GemStone currently only uses the decimalPoint setting, to provide localized reading and writing of numbers involving decimal points. Note that Smalltalk syntax requires the use of '.' as the decimal point separator, so expressions involving literal floating point numbers within Smalltalk code will still require use of the '.', regardless of Locale.

To override the operating system locale information, use the following message:

`Locale Class >> setCategory:` *categorySymbol* `locale:` *LocaleString*

The *LocaleString* passed to `setCategory:locale:` must be defined on the host machine. You can use the UNIX command **locale -a** to get a list of all available LocaleStrings.

Table F.1 lists the valid category symbols.For more information about these settings, see the operating system man page for locale.

Table F.1 Valid Category Symbols for Locale

#LC_CTYPE	Character handling
#LC_NUMERIC	Decimal point handling
#LC_TIME	Date and time handling
#LC_COLLATE	Collation setting

Table F.1 Valid Category Symbols for Locale

#LC_MONETARY	Monetary handling
#LC_MESSAGES	Messages handling
#LC_ALL	All locale categories

To use decimal localization appropriate for Germany:

```
Locale setCategory: #LC_NUMERIC locale: 'de_DE'.
```

To use UNIX default values:

```
Locale setCategory: #LC_ALL locale: 'C'.
```

The following method returns a Locale object with the locale information for the current session:

```
Locale Class >> current
```

The following method returns the decimalPoint setting for the current Locale:

```
Locale decimalPoint
```

NOTE
You can use similar methods to obtain settings for other fields, but only decimalPoint is significant in this release.

By passing in an argument of false, the following methods allow you to convert a String to a Float without consideration of the current Locale setting:

```
Float Class >> fromString:useLocale:

DecimalFloat Class >> fromString:useLocale:
```

For more information, see the image and operating system documentation for locale (man `localeconv`).

F.3 Extended Character Set Support

GemStone/S supports the ability to process and collate String data that includes extended Characters (that is, beyond the basic set of 256 Characters). This allows the various Character test, comparison, and conversion methods to work correctly across the range of Characters that can be represented in up to two bytes. You can control the contents of the character data tables that drive these methods, and can either use tables based on the Unicode Standard or design your own, based on your particular application requirements.

If you do not require more than the basic 256 Character set, you do not need to do anything. By default, GemStone uses basic 256 Character set tables, which provide the best performance. Support for Extended Character Set provides additional abilities, without compromising performance for applications that do not need them.

Extended Character Set support includes:

- Default Built-in Tables — By default, GemStone/S comes with basic character set tables for the 0–255 Character range, based on the Unicode Standard.
- 32-bit Unicode Standard — You can extend the character data tables to support the Unicode Standard through all Characters that can be represented with twobytes (0–65535).
- Flexible upgrade and customization — You can build your own character data tables, or update your applications when there are new releases of the Unicode Standard.

Extended Character Sets

GemStone Characters are based on an index, which correspond to the Unicode value. By themselves, Characters can only be compared by this index, which does not provide a good way to determine which should be sorted first, and it does not provide any information about uppercase or lowercase, or if the Character is a digit or letter.

To supply this information, GemStone provides a built-in internal table that includes the collation information for the standard 256 Characters, in a legacy collate order. This internal table provides the best possible performance for sorting or indexing of String data.

Characters that are not in the currently loaded table can still be compared and sorted, but the ordering is based on the Character value. So Character such as š that are outside the standard 256 Character range are not collated correctly by the default table. In addition, there are Characters within the standard 256 Character table that are not collated correctly for some languages; for example, ß (Character withValue: 223) by default is collated at the end of the alphabet.

If your application uses data that contains Characters such as ß and š, or any Characters beyond the standard 256 Characters, you can easily load Character Data Tables that provide collation, conversion, and querying, up to the limit of the Character set you will be using.

Character Data Tables

To hold and order the Character relationship information that is needed for collation, GemStone defines formatted Array structures referred to as Character Data Tables.

Internally, GemStone uses an primitive Character Data Table, composed of an Array of ByteArrays that contain mapping and numeric references. This format uses the minimum memory and permits the fastest access.

To allow the tables to be examined and customized, a Character Data Table can be converted into a structured form, consisting of an Array of Arrays, where each element Array contains the information for a Character.

When customized Character Data Tables are installed, either from a file or from a structured Character Data Table, the Globals Symbol Dictionary variable #CharacterDataTables is set to the primitive Character DataTable. If the variable #CharacterDataTables is nil, the default built-in 256 Character table is used. This variable is checked by each session during login, and remains in use for the lifetime of the session.

Loading Extended Character Sets

The GemStone distribution includes a file with a Character Data Table containing the complete Unicode Character Data Set, which you can load into your repository to allow you to immediately work with all Unicode Characters.

GemStone indexes and SortedCollections depend on a static collation sequence. Use caution when loading new tables. Before making changes in the Character Data Tables, all indexes on the system should be dropped, and after the new tables are installed, the indexes can be rebuilt, and SortedCollections updated.

The Character Data Tables can only be updated by SystemUser, and the changes are global for all users on a system.

To load the complete Unicode Data Set into your image:

1. Log in as SystemUser
2. execute the following:

```
Character activateCharTablesFromFile:
   '$GEMSTONE/goodies/CharTableUnicode510.dat'
```

3. Commit and log out. The new tables are in effect for subsequent logins.

While this is an easy way to load all the Character information that is available, the complete table is likely to be many more than your application actually needs.

We recommend installing customized tables containing a subset of the full Unicode Data.

However, modifying the Character Data Tables should be done rarely, as it requires index and SortedCollection rebuild. All Characters that may potentially be needed in the future by your application should be included in the customized tables.

Customizing the Character Data Table

By default, GemStone Character Data Tables are in the legacy collation sequence, to avoid problems with upgraded applications. The updated Unicode Character Data Table file, `CharTableUnicode510.dat`, includes the Unicode default collation table, DUCET. This provides a much improved default collation over the legacy collation.

However, the DUCET collation will not be correct for all national languages under all uses. GemStone allows you go customize the collation sequence if necessary.

There are several ways to customize the collation sequence.

From within GemStone, you can create a Structured Character Data Table based on the default table, or on the Unicode primitive data tables, and edit this using GemStone Smalltalk expressions. When you have modified the table to meet your needs, you can install the table into your system.

Alternatively, you can download the Unicode data and default collation table text files. After editing this raw data in text form outside of GemStone, you can use GemStone utility code to generate your own Character table .dat files, and install these files into your system.

Structured Character DataTable

The structured Character data tables allow a System Administrator to view and update the collation sequence in an easy to understand format.

A Structured Character Table is composed of an Array of elements, arranged according to character collation order. Each element is an Array containing the following entries:

- The character for this entry.
- The symbolic character category code.
- Uppercase character (if a letter) / Numerator (if numeric).

- Lowercase character (if a letter) / Denominator (if fraction).
- (Optional) Titlecase character

For example, in the default Structured Character Table, the 73rd element is the letter $a (Unicode/ASCII 97). The array at this index contains the following:

```
#( $a #Ll $A $a )
```

The first element is the Character itself; the second is the category, which in this example is lowercase, followed by the uppercase and lowercase equivalents.

To create a structured character data table based on the currently installed primitive Character Data Table, execute:

```
Character charTables
```

Once the Structured Character Data Table is correct, you can install it into your system by executing:

`Character installCharTables:` *aStructudedCharacterDataTable*

Example

Say we want to add the Characters Š (Unicode 352) and š (Unicode 353) to the standard 256-Character library. These Characters should follow S and s in collation sequence, respectively.

1. Create an instance of Structured Character Table, and place this in a temporary variable. You can generate structured Character Data Tables based on the tables currently in use using this code:

   ```
   UserGlobals at: #MyCharacterDataTables put: Character
   charTables
   ```

   In this case, since there is no customized Character Data Table already loaded, `Character charTables` returns a structured table based on the default built-in 256 Character table.

2. Add the required entries to this table. Each entry is an Array formatted as described on page 489. The entries must be positioned in the Table (Array of

Arrays) in the desired sort order, not added to the end (unless the new Character should be sorted last in any collation sequence).

```
MyCharacterDataTables
   add: (Array
         with: (Character withValue: 352)
         with: #Lu
         with: (Character withValue: 352)
         with: (Character withValue: 353))
   after: #( $S #Lu $S $s ).
MyCharacterDataTables
   add: (Array
         with: (Character withValue: 353)
         with: #Ll
         with: (Character withValue: 352)
         with: (Character withValue: 353))
   after: #( $s #Ll $S $s ).
```

3. When you are sure your changes are correct, as SystemUser, execute the following and commit:

   ```
   Character installCharTables: MyCharacterDataTables
   ```

   This converts the Structured Character Table into equivalent byte arrays and places them into the Globals variable #CharacterDataTables, to be loaded by all later sessions on login.

CAUTION
It is possible to affect or break indexes that rely on String sequencing. For this reason, we recommend you that drop any indexes based on Strings in your System before modifying the character tables.

Generating Tables from Unicode Data

To generate tables from the Unicode text files, you must load utility code that GemStone provides as an optional extension. This code is used to generate structured and primitive Character Data Tables based on the Unicode-standard raw text files.

This technique is used both for creating customized versions of the Unicode tables, as well as to rebuild the tables when a new version of the Unicode standard is released.

1. Download the Unicode data files:

   ```
   http://www.unicode.org/Public/UNIDATA/UnicodeData.txt
   ```

and save to a local file, such as `myUnicodeData.txt`.

Download the Unicode collation file:

```
http://www.unicode.org/Public/UCA/latest/allkeys.txt
```

and again, save to a local file, such as `myUnicodeCollation.txt`.

2. Optionally, edit `myUnicodeData.txt`, or `myUnicodeCollation.txt`, to modify the collation, reduce the size, or make any other changes necessary for your application.
3. Start topaz, and login as SystemUser. Load the code to build Character Data Tables

```
topaz 1> input $GEMSTONE/goodies/UnicodeData.gs
```

   and commit
4. Determine how many Characters your application needs to support and should be loaded into the tables. In this example, we will use 2048. You may be able to make this number much smaller.
5. Execute the following. This will read the data files, generate the tables, and install the tables into your system.

```
| tables|
UnicodeData
  loadFromFile: 'myUnicodeData.txt'
  max: 2048
  sort: 'myUnicodeCollation.txt'.
tables := UnicodeData generateTables.
Character installCharTables: tables.
```

6. Commit, and log out. The changes will take effect on the next login, for all users on this system.

The Unicode Database

The Unicode Consortium is an international standards organization that produces the Unicode Database, which provides unique codes for all Characters in all Character Sets. The database continues to develop; GemStone/S 6.6 uses Unicode version 5.1.

For more information on this database, refer to:

```
http://www.unicode.org/Public/UNIDATA/UCD.html
```

The Unicode Consortium provides code charts by script as well as a single master list of all characters, presented in an ASCII-only, comma-delimited version. Version 5.1 of this file has been used to create the ByteArray tables, and is provided in passivated object form.

This table can be found at

```
http://www.unicode.org/Public/UNIDATA/UnicodeData.txt
```

Character Categories

In addition to assigning unique codes for all Characters, and providing the upper and lowercase mappings, the Unicode Database also provides categories for all possible Characters. These categories allow letters to be distinguished from numeric characters and from punctuation.

Table F.2 lists all the Unicode categories. While some of these Character categories are commonly used, other categories are more rare and are not used in English or Latin-based languages.

Table F.2 Character Category Codes and Symbols

1	`#Lu`	`Letter, Uppercase`
2	`#Ll`	`Letter, Lowercase`
3	`#Lt`	`Letter, Titlecase`
4	`#Lm`	`Letter, Modifier`
5	`#Lo`	`Letter, Other`
6	`#Mn`	`Mark, Nonspacing`
7	`#Mc`	`Mark, Spacing Combining`
8	`#Me`	`Mark, Enclosing`
9	`#Nd`	`Number, Decimal Digit`
10	`#Nl`	`Number, Letter`
11	`#No`	`Number, Other`
12	`#Pc`	`Punctuation, Connector`
13	`#Pd`	`Punctuation, Dash`
14	`#Ps`	`Punctuation, Open/Start`
15	`#Pe`	`Punctuation, Close/End`
16	`#Pi`	`Punctuation, Initial Quote`
17	`#Pf`	`Punctuation, Final Quote`

Table F.2 Character Category Codes and Symbols (Continued)

18	`#Po`	`Punctuation, Other`
19	`#Sm`	`Symbol, Math`
20	`#Sc`	`Symbol, Currency`
21	`#Sk`	`Symbol, Modifier`
22	`#So`	`Symbol, Other`
23	`#Zs`	`Separator, Space`
24	`#Zl`	`Separator, Line`
25	`#Zp`	`Separator, Paragraph`
26	`#Cc`	`Other, Control`
27	`#Cf`	`Other, Format`
28	`#Cs`	`Other, Surrogate`
29	`#Co`	`Other, Private Use`
30	`#Cn`	`Other, Not Assigned`

Categories are used to answer these queries:

```
Character >> isDigit
Character >> isLetter
Character >> isLowercase
Character >> isUppercase
Character >> isSeparator
```

Titlecase category

In addition to uppercase and lowercase, the Unicode Standard provides the case-related category Titlecase (`#Lt`). This is a special case for composite characters that incorporate multiple individual characters—for example, the Unicode Character code 0x01C5 (decimal 453), DZ. (Such characters are rare; there are only 12 such characters in the Unicode Standard.) Titlecase of these composite character forms consists of an Uppercase first character, followed by all remaining characters in lowercase. In this example, the uppercase would be DŽ, the lowercase would be dž, and the titlecase Dž.

For most letters, Titlecase is the same as Uppercase. Titlecase is an optional feature in GemStone's formatted tables.

GemStone Character Table Data Files

The $GEMSTONE/goodies directory contains two files in GemStone passivate format:

`CharTableDefault.tab`
: Character table data for the default table, in structured form, in legacy collate order.

`CharTableUnicode410.dat`
: Character table data for 16-bit Unicode, in low-level ByteArray form, in legacy collate order. You can use this file to install the extended Unicode tables. (See "Loading Extended Character Sets" on page 488.)

`CharTableUnicode510.dat`
: Character table data for 16-bit Unicode, in low-level ByteArray form, in DUCET collate order. (See "Loading Extended Character Sets" on page 488.)

`UnicodeData.gs`
: This includes the GemStone Smalltalk source code to generate Character Data Tables based on the raw Unicode text files.

Archiving and distributing Character Data Tables

Character Data Tables can be stored in operating system files in passive object format, and loaded from these passivated files. This allows you to archive and distribute the specific Character sets and collation your application is using.

To save the current contents of the Globals variable #CharacterDataTables to a passivated file, execute:

```
Character passivateCharTablesToFile: 'MyFileName'
```

To install a previously passivated from a file, log in as System user and execute:

```
Character activateCharTablesFromFile: 'MyFileName'
```

Commit and log out.

Troubleshooting

Restoring the default table

If you want to clear out the Character DataTables, and return to using the default built-in 256 Character tables, do the following:

1. Log in as SystemUser.
2. Execute:

   ```
   Globals removeKey: #CharacterDataTables.
   ```

Commit, and log out.

Disabling load of the Character Data Table

When Character Data Tables are loaded during login, GemStone can sometimes detect if an invalid Character Data Table is installed and will revert back to the default 256 character data table. But in other cases, invalid tables may cause problems interpreting Smalltalk source code or topaz commands. In these cases you may need to manually disable the loading of the character data table.

To recover from serious problems after installing an invalid character data table, do the following:

1. At the operating system level, set the host environment variable GS_DISABLE_CHARACTER_TABLE_LOAD to TRUE. The particular value of this environment variable does not matter; its existence is the critical factor.
2. Start a new Topaz session and log in as SystemUser.
3. Execute the following:

   ```
   Globals at: #CharacterDataTables put: nil.
   ```

4. Commit.
5. Use caution when using this environment variable. Like any use of a different collation sequence, data structures that depend on collation, such as indexes, will not work correctly and can become inconsistent if modified.

Appendix G 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 (page 497)
- Statmonitor command line options (page 509)
- VSD menu reference (page 510)
- Descriptions of VSD files, including the syntax for template filters (page 512)

G.1 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 `markForCollection` 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 stand-alone 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 independent of the operating system platform, version and server product that created the statmonitor flat file. However, older versions of VSD cannot always read statmonitor files produced by more recent GemStone server versions.

  On UNIX platforms, VSD requires X-Windows.

statmonitor 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:

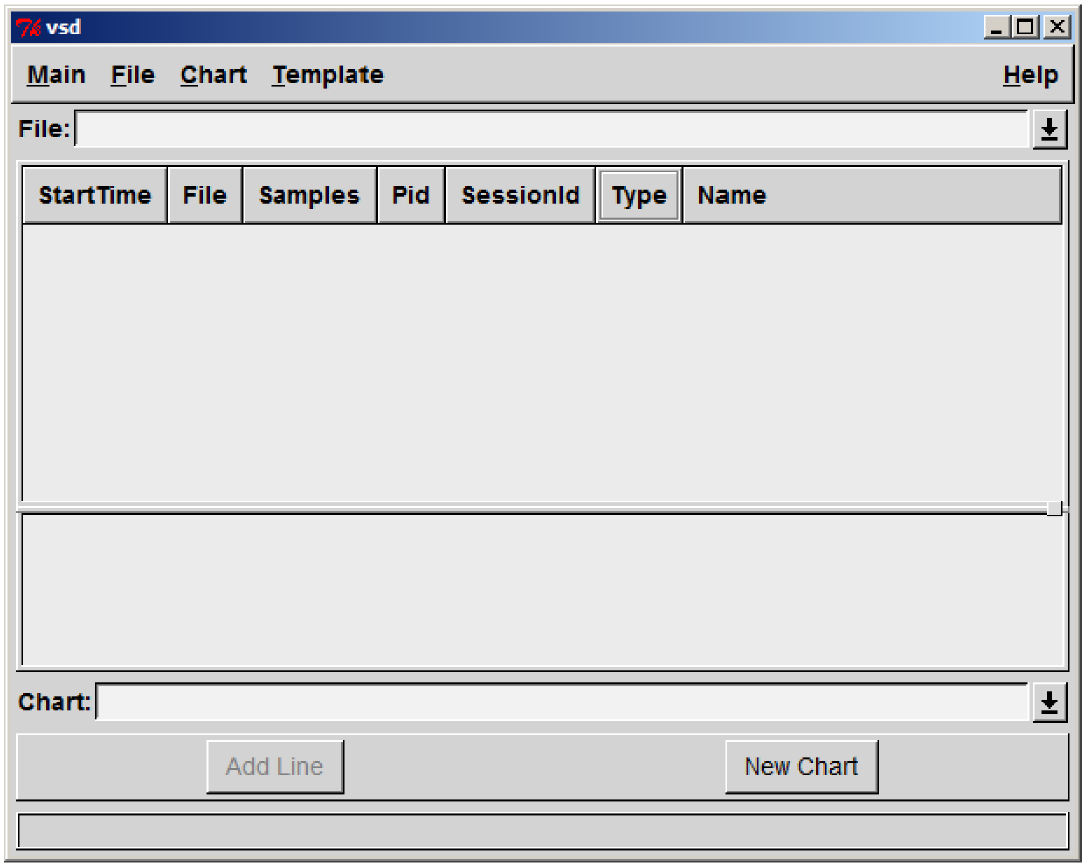
`statmonitor` *stoneName* `-f` *outputFile* `-i` *sampleInterval*

For complete statmonitor command line syntax, see “Statmonitor command line syntax” on page 509.

To start VSD, simply execute the `vsd` command. (This assumes that `$GEMSTONE/bin` defined in your path.)

The initial startup screen will appear similar to that shown in Figure G.1.

Figure G.1 VSD Startup Screen



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.
- If you know the name of the load file, you can type the full path in the File entry box, 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.

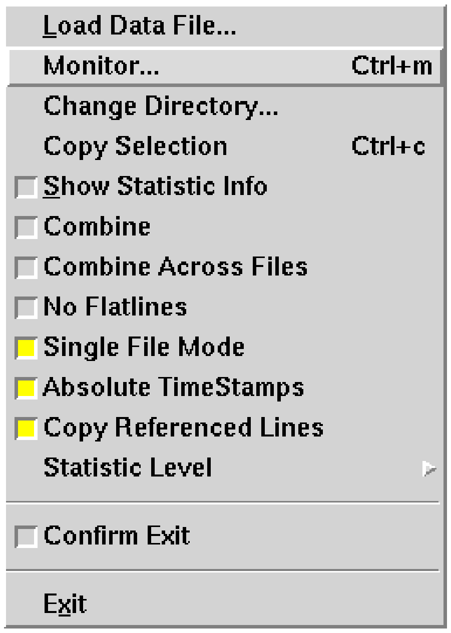
Maintaining a current view of the data file

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.

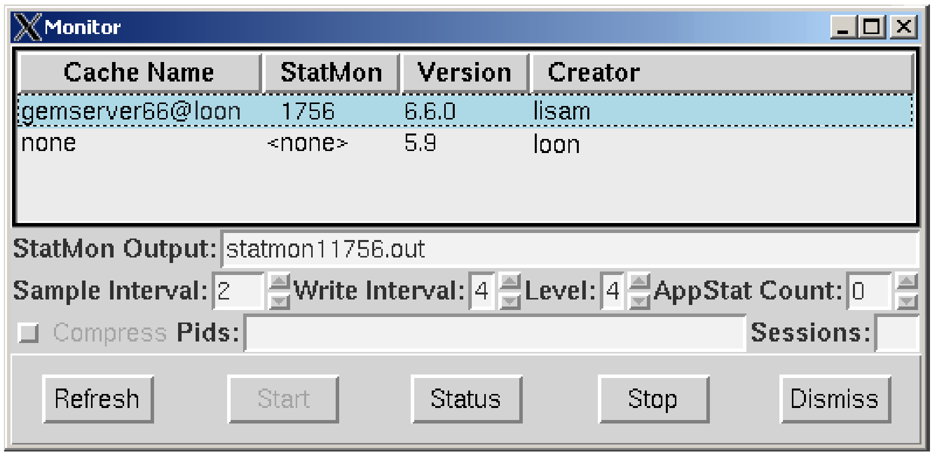
Starting statmonitor from VSD and viewing current data

You can also use VSD to start statmonitor and read directly from the data file statmon is currently creating. To do so:

Step 1. Choose **Monitor** from the **Main** menu.



Step 2. When the Monitor window appears, click to select the cache on the local machine for which you want to obtain statistics.



Cache Name	StatMon	Version	Creator
gemserver66@loon	1756	6.6.0	lisam
none	<none>	5.9	loon

StatMon Output: statmon11756.out
Sample Interval: 2 Write Interval: 4 Level: 4 AppStat Count: 0
Compress Pids: Sessions:

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 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.

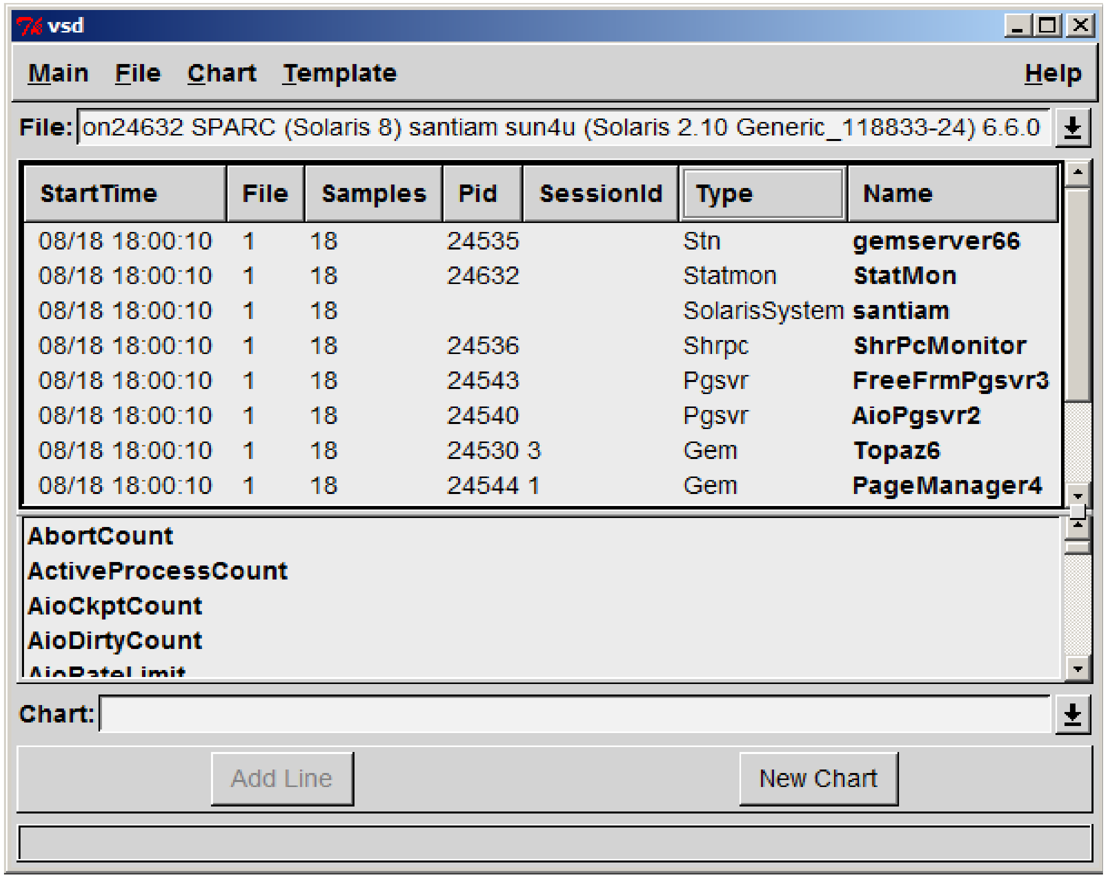
Step 4. When you're done, click **Start**.

VSD loads the data file as soon as it has some samples in it (unless you have explicitly turned off Auto Update).

Step 5. When you're finished collecting statistics, click **Stop**.

Viewing Statistics

After you've loaded the data file or started monitoring, the VSD window looks something like this:



File: on24632 SPARC (Solaris 8) santiam sun4u (Solaris 2.10 Generic_118833-24) 6.6.0

StartTime	File	Samples	Pid	SessionId	Type	Name
08/18 18:00:10	1	18	24535		Stn	gemserver66
08/18 18:00:10	1	18	24632		Statmon	StatMon
08/18 18:00:10	1	18			SolarisSystem	santiam
08/18 18:00:10	1	18	24536		Shrpc	ShrPcMonitor
08/18 18:00:10	1	18	24543		Pgsvr	FreeFrmPgsvr3
08/18 18:00:10	1	18	24540		Pgsvr	AioPgsvr2
08/18 18:00:10	1	18	24530	3	Gem	Topaz6
08/18 18:00:10	1	18	24544	1	Gem	PageManager4

AbortCount
ActiveProcessCount
AioCkptCount
AioDirtyCount
AioRateLimit

If you are monitoring, the green numbers in the Samples column indicate processes that are still running. The black numbers indicate processes that have stopped.

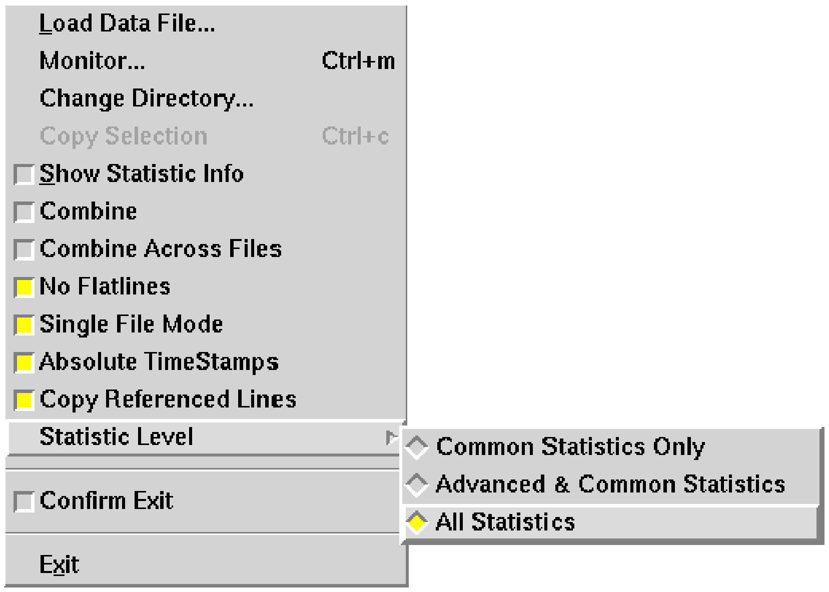
Your next step is to select the specific cache statistics to view. Statmonitor keeps statistics for various GemStone processes, corresponding to the cache slots described on page 237. They appear as follows under the heading **Type** in the process list:

Stn	Stone
Gem	Gem
Shrpc	shared page cache monitor
Pgsvr	AIO or free frame page server
Statmon	statmonitor
Appstat	application-defined statistics

NOTE
For detailed descriptions of each statistic, see "Cache Statistics" on page 239.

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, click the left mouse button on the process(es) you wish to view.

You can also use the right mouse button popup 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.

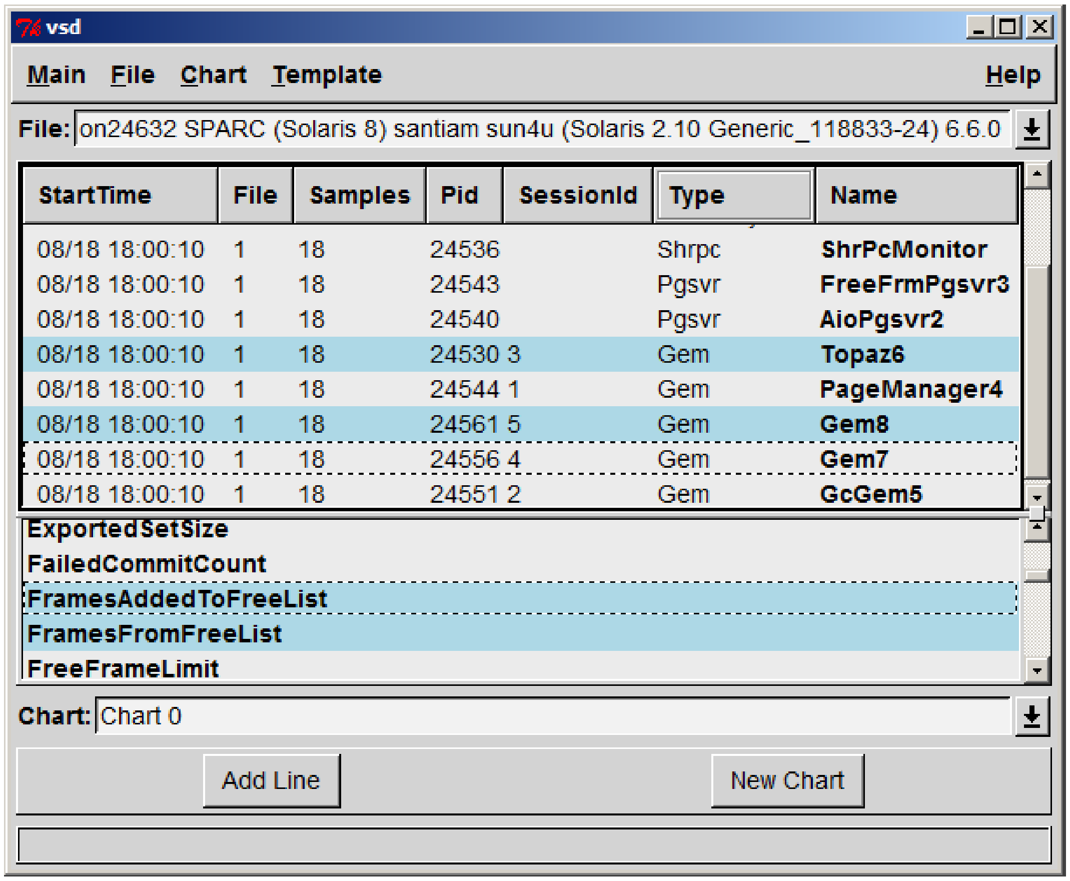
  To select the highlighted item, click on it. When you're done, press Return.

- Select all processes, all Gem processes, or all page server processes.
- Combine multiple processes into a single line.

  This can be quite helpful. For example, if you want to measure page reads per second for several hundred Gem processes, it's not possible to fit all the individual lines and legends onto one chart. If you select all Gem processes, then combine them into a single line in the chart, the data is easily readable.

- Eliminate *flatlines*—processes whose values are always zero.

Figure G.2 Process List with Selected Gems



File: on24632 SPARC (Solaris 8) santiam sun4u (Solaris 2.10 Generic_118833-24) 6.6.0

StartTime	File	Samples	Pid	SessionId	Type	Name
08/18 18:00:10	1	18	24536		Shrpc	ShrPcMonitor
08/18 18:00:10	1	18	24543		Pgsvr	FreeFrmPgsvr3
08/18 18:00:10	1	18	24540		Pgsvr	AioPgsvr2
08/18 18:00:10	1	18	24530	3	Gem	Topaz6
08/18 18:00:10	1	18	24544	1	Gem	PageManager4
08/18 18:00:10	1	18	24561	5	Gem	Gem8
08/18 18:00:10	1	18	24556	4	Gem	Gem7
08/18 18:00:10	1	18	24551	2	Gem	GcGem5

ExportedSetSize
FailedCommitCount
FramesAddedToFreeList
FramesFromFreeList
FreeFrameLimit

Chart: Chart 0

Step 3. Next, select one or more statistics in the scrolling list of available statistics for the selected processes. You can also type a letter to reach the first statistic that starts with that letter.

If you have selected processes of different types, only the statistics that are available for each of the processes will be displayed. Statistics that contain non-zero data points are shown in bold.

Step 4. After you've selected the statistic, do one of the following:

- To display the statistic in a new chart, click **New Chart** button. To explicitly name the chart, enter a name in the Chart entry field.
- 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**. If you do not select a chart, the statistics will be displayed in the most recently selected chart.

Figure G.3 Chart Displaying Statistics

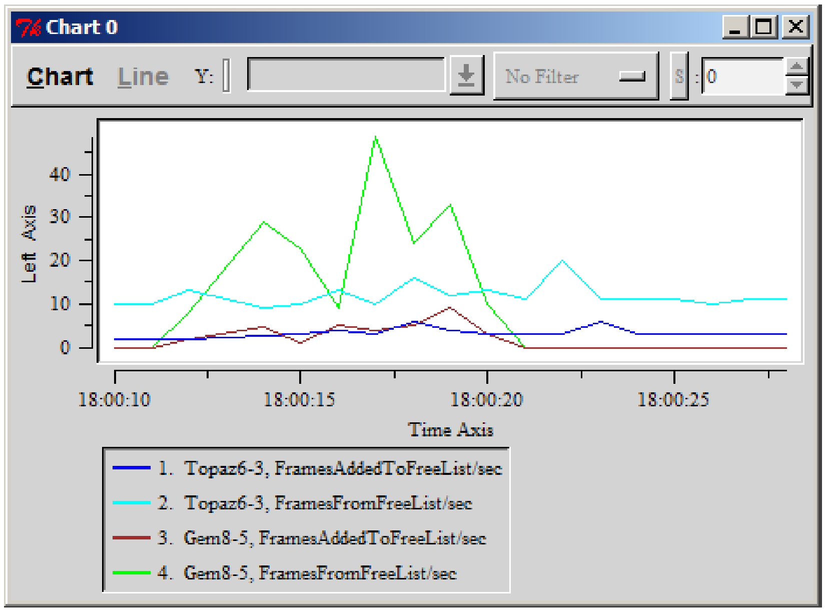


Chart 0
Chart
Line
Y:
No Filter
0
Left Axis
0
10
20
30
40
18:00:10
18:00:15
18:00:20
18:00:25
Time Axis
1. Topaz6-3, FramesAddedToFreeList/sec
2. Topaz6-3, FramesFromFreeList/sec
3. Gem8-5, FramesAddedToFreeList/sec
4. Gem8-5, FramesFromFreeList/sec

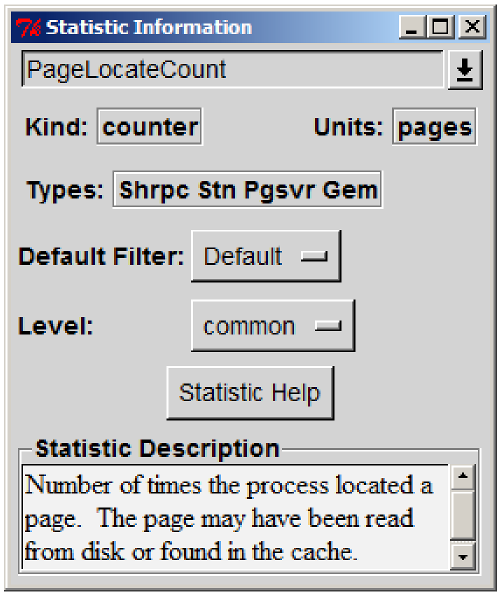
Step 5. To add another statistic to the chart, repeat steps 3 and 4

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. The line will be highlighted in red.
- 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 Statistics Info** from the **Main** menu. You'll see a Statistic Information window that looks something like this:



Number of times the process located a page. The page may have been read from disk or found in the cache.

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:

```
System _cacheName: 'myName'
```

Filter

Whenever you add a line to a chart, the filter determines how much information is displayed for the selected 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 512.*

To do this...	Do this...
Create a new chart from a template	In the VSD main window, choose **New from Template** from the **Chart** menu. This is a good way to display some of the more useful system statistics.
Apply a template to the chart that you're viewing	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.*
Reread the template file `.vsdtemplates` into VSD after you've edited it	In the VSD main window, choose **Reload Template File** from the **Chart** menu.
Save the current chart as a template	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.

G.2 Statmonitor command line syntax

statmonitor *stonename* **-f** *fileName* [*options*]

-f *fileName*	The output filename. By default, the output filename is statmon*N*.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); the 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*	The level of system statistics to collect. The system statistics that are collected is operating system-specific. The default *level* is 1. When *level* is 0, no system statistics are collected. For details on the current system statistics collected, execute `statmonitor -h` at the command line.
-S	Sample only the stone and shared cache monitor.
-P	Sample the stone, shared cache monitor, and all AIO page servers only.

G.3 VSD Menu Reference

Chart menu

To customize the way VSD displays statistics in your chart, choose items from the Chart window's **Chart** menu.

- **Zoom In** — You can zoom to improve your view of the chart. After you choose this menu item, click to select one corner of the area that you want to zoom.

Move the mouse pointer to the opposite corner of the zoom area, then click again.

To quickly zoom in on an area, move the mouse pointer over the area and click the middle mouse button (if available).

- **Zoom Out** — Select this menu item or click the right mouse button to reverse the effect of one zoom-in operation.
- **Compute Scale All, Unscale All** — To view multiple statistics on the same axis, it's sometimes helpful to adjust the scale of the chart.
- **Show Legend** — Select this item to display the legend for this chart.
- **Time Format** — Changes the format of the time displayed along the X axis.
- **Show Time Axis Title, Show Left Axis Title, Show Right Axis Title** — Select these items to display the title alongside the respective axes.
- **Show Current Values** — Select this item to display the current X and Y values for the selected line at the top of the chart.
- **Show Min and Max** — Select this item to display the minimum and maximum values for the selected line at the top of the chart.
- **Show Line Stats** — Select this item to display the following statistics for the selected line: the number of data samples, the minimum, the maximum, the mean, and the standard deviation. The statistics are calculated from all of the data points on the selected line in the region defined by the graph's current X axis. (To change the region, use the **Zoom In** or **Zoom Out** menu item.)
- **Show CrossHairs** — Select this item to display crosshairs centered at the current cursor position, useful to precisely locate a point of interest in time or on the vertical scaled.
- **Show Gridlines** — Select this item to display gridlines in the chart background, also useful for precisely locating a point of interest.

For additional information about what you can do in the Chart window, choose **Help** from the **Chart** menu.

Line menu

Line menu commands operate on the currently selected line. To select a line, click on it or on its entry in the chart legend. VSD highlights the selected line.

- **Log Info** — Displays a log file showing the line statistics for all data samples in the region defined by the graph's current X axis.

- **Log Delta** — Use this menu item to measure the difference between two values on the selected line.

  Select the line, choose this menu item, then click on the two points whose difference you want to compute. VSD responds by displaying a log file showing the difference in time and value between the two points; the number of data samples in the selected line segment; and the minimum, maximum, mean, and standard deviation of those samples.
- **Compute Scale** — Computes a scale value for the selected line o make it visible on the current chart. You can also use the Scale entry box to manually change the scale. The default scale value is 1.
- **Unscale** — Reverses the effect of **Compute Scale**.
- **Graph on Left Axis** — Select this item to display the Y axis for the selected line to the left of the chart. Otherwise, the Y axis is displayed to the right.

  To view multiple lines on the same chart, you can use this menu item to graph large values on one axis and small values on the opposite axis.
- **Symbol** — Selects a new symbol.
- **Update** — Updates the selected line, assuming that values are being monitored in real time and that **Auto Update** is turned off on the main window's File menu.
- **Delete** — Removes the selected line from this chart.

G.4 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 named VSD chart. For example, the following is a template for displaying a chart with information that you can use to determine if your Shared Page Cache is too small:

```
set vsdtemplates(CacheTooSmall) {
        {Shrpc ShrPcMonitor FreeFrameCount 1 none y2}
        {+ {*} FramesFromFreeList 1 persecond y}
        {+ {*} FramesFromFindFree 1 persecond y}
    }
```

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 counter 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}
```

For more information on VSD templates and the syntax and options available, see the VSD help under the topic "Template Syntax".

Appendix

H

Object State Change Tracking

Introduction

GemStone transaction logs (tranlogs), which provide a way to recover all changes made to the repository in the case of repository crashes, or allow warm standbys to apply changes made to a primary repository, include detailed records of all changes in an encoded and compressed form. Converting the tranlog information to human-readable form, and analyzing this output, provides invaluable information for debugging and testing. It allows us to determine exactly what changes have been made to any of the objects in the repository over time. The tools used to perform this have been used internally at GemStone for many years, and have been provided to customers on occasion when needed to analyze specific application problems.

The ability to track changes to objects may be useful for customers who need to identify details on changes that have been made to application data objects. For this reason, these scripts are available as part of the GemStone/S product. Additional information has been added to the tranlogs to allow tracking of the specific user or machine that originated the object changes.

NOTE
To provide real-time tracking of changes to the repository, you must set STN_TRAN_FULL_LOGGING to True.

The following tranlog analysis scripts are located in the `$GEMSTONE/bin/` directory:

`printlogs.sh` — Output the complete contents (optionally filtered) of selected tranlogs in human-readable form.

`searchlogs.sh` — Search all tranlogs in a directory for selected OOPs. Output the matching entries (optionally filtered) in human readable form.

A GemStone repository performs many automatic operations that are transparent to end users — for example, garbage collection and checkpoints. The tranlogs, of course, must contain records of any changes made by these operations. Complete details on everything that tranlogs may contain is beyond the scope of this documentation. The information provided here is intended to allow the use of the tranlog analysis scripts to locate and identify the details of changes to application objects.

In object-oriented design, the principle of *encapsulation* allows you to hide the internal complexity of objects. However, to understand the data recorded in the tranlogs, you must have a detailed understanding of the actual structure of the objects. This includes both your own application classes, and the classes that are part of GemStone Smalltalk.

Finally, keep in mind that the tranlog analysis scripts are general purpose, used for a wide variety of purposes in which a detailed record of internal repository operation is required. For this reason, the scripts may output much more information than is necessary for tracking object state changes. You may need to ignore this extraneous information as you perform your analysis.

Tranlog Analysis Scripts

Prerequisites

To use the tranlog analysis scripts, the environment variable `$GEMSTONE` must be set, and the `$GEMSTONE/bin/` directory must be in your executable path.

The scripts perform analysis on tranlogs in the current working directory. If you are using raw partitions for your tranlogs, locate disks on the file system with adequate space both for the tranlogs themselves, and for the script output files — which may be larger than the original tranlogs. Use `copydbf` to copy the tranlogs from the raw partition to the file system. For more information on the `copydbf` utility, see Appendix B, "GemStone Utility Commands.".

Output

Output from the scripts goes directly to `stdout`. To preserve the output and allow it to be used in later steps of analysis, redirect it to a file. For example:

```
$> printlogs.sh tranlog1.dbf > tranlog1.out
```

Because these scripts can produce very large amounts of output, make sure that you have adequate disk space.

In some cases, the resulting files may be too large to open with a UNIX text editor such as `vi` or `emacs`. You may find it necessary to use the UNIX `split` utility to break up very large output files into more manageable chunks.

Assumptions

By default, the scripts assume that the tranlogs are named using the GemStone convention `tranlog`*NNN*`.dbf`. If you are using a different naming convention, you can override this by setting the environment variable `$GS_TRANLOG_PREFIX`.

The scripts use the tranlog file's creation date to determine the order in which the tranlogs are analyzed. If you copy or manipulate the tranlogs in a way that changes the creation date, this may cause the analysis to be done out of order. The output will warn of this with the message:

```
*** Warning:  scanning tranlogs out of order
```

Filter Criteria

Both scripts allow you to filter the results, so that you can locate entries that are related to a particular UserId, GemHost, or ClientIP. You can include the following command-line options to specify search criteria:

`user` <*userId*> — Filter by the userId (the user name) of the GemStone **UserId** – The userId (user name) of the UserProfile associated with this session: DataCurator, SystemUser, etc. The filter keyword is `user`.

GemHost – The name or IP address of the host machine running the gem process. For a linked session, which links the gem into the client, this is the same machine as the client.

If the gem is running on the same machine as the stone, use the name of the host machine. Otherwise, if the gem is on a different machine than the stone, use the IP address of the remote machine.

The filter keyword is `host`.

ClientIP – The IP address of the host machine running the client process.

For an RPC session, this is the machine running the client application. Clients may be topaz -r, GemBuilder for Smalltalk, or GemBuilder for C applications. For a linked session, this is the machine running the linked client/gem (the same machine as the GemHost). However, the ClientIP is always the IP address, even if it is on the same machine as the stone.

The filter keyword is `client`.

Effective UNIX user ID – The integer that is the effective UNIX user id of the gem process.

The filter keyword is `euid`.

UNIX user ID – The integer that is the real UNIX user ID of the gem process.

The filter keyword is `ruid`.

effective UNIX user name – The effective UNIX user name of the gem process.

The filter keyword is `euidstr`.

UNIX user name – The real UNIX user name running the gem process.

The filter keyword is `ruidstr`.

process ID – The integer process id (PID) of the gem process.

The filter keyword is `gempid`.

session ID – The integer session id of the session within GemStone.

The filter keyword is `sessionid`.

NOTE
*Information about UserId, GemHost, and ClientIP are derived from a **Login** tranlog entry created when a session first logs in. This entry associates the UserId/GemHost/ClientIP with a particular sessionID, which is then used as a key for subsequent tranlog entries. If you start analysis from a subsequent tranlog that does not include this **Login** entry, these fields will be left blank for that session, and printouts/searches using filters based on these fields will not locate any results. Likewise, scanning through tranlogs out of order may result in the wrong **Login** entry being associated with a given sessionID. This*

would set UserId/GemHost/ ClientIP incorrectly for that particular session, and would produce incorrect results when filtering.

printlogs.sh

This script prints out the contents of one or more tranlogs in the current working directory in a human-readable form.

NOTE
This script produces a very large amount of output, which (unfiltered) will exceed the size of original tranlog(s), and may be up to twice as large as the original tranlogs. Before running this script, consider disk space, the use of filters, and restricting the set of tranlogs. Use caution in including the `full` *keyword.*

Usage

`printlogs.sh [<`*filters*`>] [full] [all] [<`*tlogA*`> ... <`*tlogZ*`>]`

<*filters*> — If you specify command-line filters, the script only prints out the tranlog entries that match the specified criteria. Filters may be combined. For details, see "Filter Criteria" on page 517.

`full` — Produce more detailed information. (Keep in mind that this produces *much* larger output results.)

`all` — Print out the contents of all tranlogs in the current working directory.

Examples

To print out the entire contents of all tranlogs in this working directory:

```
printlogs.sh all
```

To print out all entries in a selected number of tranlogs (note that tranlogs in the sequence must be contiguous):

```
printlogs.sh tranlog5.dbf tranlog6.dbf tranlog7.dbf
```

To print out all tranlog entries for the user DataCurator in any tranlog:

```
printlogs.sh user DataCurator all
```

To print out detailed information for all entries in `tranlog5.dbf` for the user DataCurator:

```
printlogs.sh full user DataCurator tranlog5.dbf
```

searchlogs.sh

This script prints out tranlog entries associated with specified OOP values, according to the command line arguments. All tranlogs in the current working directory are scanned.

Usage

searchlogs.sh [<*filters*>] <*oopA*> ... <*oopB*>]

<*filters*> — If you specify command-line filters, the script only prints out the tranlog entries that match the specified criteria. Filters may be combined. For details, see "Filter Criteria" on page 517.

When using more than one filter, the fileters must be in the listed order.

Examples

To print out all entries involving OOP 1234 and OOP 5678:

```
searchlogs.sh 1234 5678
```

To print out all entries involving OOP 1234 performed by DataCurator:

```
searchlogs.sh user DataCurator 1234
```

To print out all entries involving OOP 1234 and OOP 5678 performed by DataCurator while logged in from client machine 10.20.30.40:

```
searchlogs.sh user DataCurator client 10.20.30.40 1234
5678
```

Tranlog Structure

In order to make sense of the tranlog script output, you need a basic understanding of how tranlogs are structured.

GemStone transaction logs consist of a sequence of *tranlog records*. Tranlog records are written on *physical pages* of 512 bytes (note that this is different from the larger page size used for extents). A tranlog record may extend over more than one page, but its size is always a multiple of 512 bytes.

Each tranlog record contains one or more *tranlog entries* (sometimes referred to internally as logical records). The tranlog entries contain the information of interest — the actual changes to objects in the repository (and any other repository operations).

Output from the scripts includes header information for the tranlog record (see Example H.1), followed by data from each of the tranlog entries held by that tranlog record.

Example H.1 Tranlog Record Header Output

```
Dump of page 3, Kind=(16)Tran Log Record
  thisRecordId:(file:2 rec:3) previousRecordId:(file:2 rec:2)
  recordSize: 512 numLogicalRecs: 1 fileCreationTime: 1156195399
```

A tranlog record is identified by the pageId on which it begins. Since a tranlog record may extend over multiple pages, there may be a gap in the sequence of pageIds in the output. For example, if the tranlog record starting on page 6 has a recordSize of 1024 (two 512-byte physical pages), then the pageId of the next tranlog record will be 8. See Example H.2.

Example H.2 Gap in Tranlog Record Sequence

```
Dump of page 6, Kind=(16)Tran Log Record
 thisRecordId:(file:3 rec:6) previousRecordId:(file:3 rec:5)
 recordSize: 1024 numLogicalRecs: 1 fileCreationTime: 1156738357


3.6.0  Login session: 5 user: DataCurator gemhost: myhost clientIP:
10.20.30.40 beginId:(26750 0)

---------------------------------------
Dump of page 8, Kind=(16)Tran Log Record
 thisRecordId:(file:3 rec:8) previousRecordId:(file:3 rec:6)
 recordSize: 512 numLogicalRecs: 1 fileCreationTime: 1156738357


3.8.0  Checkpoint session: 0 beginId:(26750 0)
  rootPageId: 3
  preChkLogRecordId: (file:3 rec:6) stoneStartup: 0
  logDebugLevel: 0  spare3..4: 0 0
```

Tranlog Entries

Each tranlog entry contains a unique identifying code, a descriptive entry type, and information on the session that originated the tranlog entry.

The identifying code consists of three numbers in the form:

```
AAA.BBB.CCC
```

where:

`AAA` – the fileId of the tranlog holding the entry
`BBB` – the pageId for the tranlog record holding the entry
`CCC` – the entryId: the number of the entry within the tranlog record

Each tranlog entry is of a particular type, according to the event that it represents and the information it contains. Types are indicated by short descriptive terms such as `Login`, `Abort`, `Commit`, and `Data`. There are a large number of entry types, most of which are not important for tracking object state changes and can be ignored (for example, the `Checkpoint` entry in Example H.2). The next section lists the entry types that are important.

Each tranlog entry is associated with an Integer, the sessionID. A sessionID is unique to a specific session at any point in time. However, when a session logs out, the sessionID becomes available again for reuse by a new session logging in, so over a period of time, the same sessionId may be used by a number of different sessions.

A sessionID of zero is used for tranlog entries generated by the Stone. (See the `Checkpoint` entry in Example H.2.)

If the entry was not originated from the Stone (that is, the tranlog entry's sessionID is not 0), the tranlog entry header includes more information about the session: the UserId, the GemHost, and the ClientIP address. These are described in more detail under "Filter Criteria" on page 517.

Example H.3 Example Tranlog Entry

```
2.4.1  StartGcGem session: 1 user: GcUser gemhost: myhost clientIP:
10.20.30.40 beginId:(67 3)
```

Example H.3 shows that it is in the tranlog with fileId 2 (by the default naming convention, `tranlog2.dbf`), it is in the fourth physical page and in tranlog record 4, and it is the first tranlog entry in tranlog record 4.

This entry is of the tranlog entry type `StartGcGem`. The session is logged in as the user named `GcUser`; the Gem session is executing on the same machine as the Stone, a machine named `myhost`; and the client is executing on a machine with the IP address `10.20.30.40`. (beginId contains transaction tracking information that you can ignore.)

Tranlog Entry Types

There are a large number of tranlog entry types. Most of these are not relevant to tracking object state history; they record internal operations of the system, such as garbage collection or checkpoints. Such tranlog entry types are not documented, although their functions can often be deduced by their names.

The following entries are most important for tracking object state history.

Login

A new session is logging into GemStone. As mentioned earlier, this entry sets the UserId, GemHost, and ClientIP data for this particular sessionID. If you start analysis on a tranlog that follows this event, these fields will be left blank for that session.

For example, if session 7 logs in while tranlog4 is in effect, and logs out when tranlog5 is in effect, and you begin analysis on tranlog5, entries for this session will not contain any UserId/GemHost/ClientIP information. If sessionID 7 is reused by a new session logging in later during tranlog5, that login will be recorded in tranlog5, and subsequent entries for this new session *will* be displayed properly.

There is no tranlog entry recorded when a session logs out.

BeginData
Data
BeginStoreData
StoreData

GemStone uses these four entry types for recording new or changed object data. The basic entry information is followed by additional information about the changes, including the OOP values of the changed objects. If you use the optional "full" flag in the `printlogs.sh` script, the output includes additional information on the exact changes made.

Commit

All the changes recorded in earlier `BeginData`, `Data`, `BeginStoreData`, or `StoreData` entries are now officially committed to the repository.

Abort

Any changes recorded earlier in this transaction are discarded.

BreakSerialization

This entry indicates that a transaction conflict occurred during an attempt to commit. Any changes recorded earlier in this transaction are not yet permanent. This event is usually followed by an `Abort` entry, although it's possible that the next event seen for this sessionID is a `Login`, if the original session logged out and a new session reuses the sessionID.

Very Large Objects

GemStone is designed so that all objects will fit on a single page of 8192 bytes. This means that a byte object can be no larger than about 8000 bytes (since page header information uses some space), and pointer objects can only have about 2000 elements. GemStone internally represents objects larger than this as a tree structure, where the root node object references two or more leaf node objects, which then reference the actual elements of the collection object. Extremely large objects, such as large collections, may have internal branch nodes, if the number of leaf objects needed exceeds 2000.

This internal structure is usually transparent to the user. So, for example, you may create and manipulate an Array containing 10,000 elements as if it was a single large object, while the actual representation is a root object that references five leaf objects, each containing a 2000-element chunk of the Array. While this makes application development with GemStone much simpler, the entries in the tranlogs reflect the actual implementation; you need to be aware of this to understand tranlog output relating to collections larger than ~2000 object references or ~8000 bytes. Adding an element to the large Array, for example, may mean that the tranlog entry includes a change to an instance of LargeObjectNode (the leaf node object), rather than a change to the Array itself.

Full vs. Normal Mode

When using the `printlogs.sh` script, you can optionally specify “full” mode to get more details on the changes made to objects in the repository. But this greatly increases the size of the resulting log files. For example, using normal mode on our test tranlogs generated a log file that contained an entry that looked like this:

Example H.4 Tranlog Entry, Normal Mode

```
5.7.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(151 2)

            clusterId: 1,  extentId: 0  numObjs:5
      154297  155069  155125  155185  155245
```

This entry tells you that five objects were created or modified during this event, with OOPs 154297, 155069, 155125, 155185, and 155245.

Using “full” mode produces a much more detailed listing for the same event. See Example H.5.

Example H.5 Tranlog Entry, Full Mode

```
5.7.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(151 2)
            clusterId: 1,  extentId: 0  numObjs:5

objId: 154297 class: 1741 seg: 3253 sz: 14 bits: 0x27 psize: 76
Oop values:  19  7  11  23  10  154177  7845  155245
8: 10  10  10  10 78277  154233

objId: 155069 class: 182393 seg: 3253 sz: 5 bits: 0x67 psize: 40
Oop values:  155185  10  155125  156661  389295

objId: 155125 class: 1169 seg: 3253 sz: 8 bits: 0x3007 psize: 28
Bytes:  Oakville

objId: 155185 class: 1169 seg: 3253 sz: 16 bits: 0x3007 psize: 36
Bytes:  5234 Main Street

objId: 155245 class: 1745 seg: 3253 sz: 2 bits: 0x7 psize: 28
Oop values:  7845  155053
```

In full mode, the entry includes a description of every object that was created or modified, containing these fields:

`objId`: The OOP of this object.

`class`: The OOP of the class of this object.

`seg`: The OOP of the segment associated with this object.

`sz`: The logical size of this object (in bytes for a byte object, OOPs for an OOP object).

`bits`: The format bits for this object (internal GemStone use).

`psize`: The physical size of this object on disk in bytes (including 20 bytes of object header information).

`Bytes`: The actual bytes that make up this object (if a byte object).
`Oop values`: The actual OOPs that make up this object (if an OOP object).

If the OOP values contain more than eight elements, they are broken into lines of eight items, each of which is prefixed by a counter. For example, an Array of 30 items might look like this:

```
objId: 210841 class: 1045 seg: 3261 sz: 50 bits: 0x47 psize:
220
Oop values:  10 10 210829  210825  210821  210817  210813
210809
8:  210805  210801  210797  210793  210789  210785  210781
210777
16:  210773  210769  210765  210761  210757  210753  210749
210745
24:  210741  210737  210733  210729  210725  210721
```

Bytes are broken up similarly, but the sections are 60 bytes rather than 8. For example, the source code string for the `name:` method may look like this:

```
objId: 141141 class: 1169 seg: 3261 sz: 91 bits: 0x3007
psize: 112
Bytes:  name: newValue

  "Modify the value of the instance variabl
61: e 'name'."
name := newValue
^@
```

Tranlog Analysis Example

This example is based on a simple database that contains some Employee information. The tranlog reflects some simple operations, creating and modifying these Employee objects.

The structure of classes associated with the Employee data is as follows:

Employee:

name (a Name object)
age (a SmallInteger)
address (an Address object)

Name:

last (a String object)
first (a String object)
middle (a String object)

Address:

addr1(a String object)
addr2 (a String object)
city (a String object)
state (a String object)
zip (a SmallInteger)

To begin, User1 creates five Employee objects (with associated Name and Address objects). User1 and User2 subsequently make some minor changes to one of the Employees:

- User2 incremented the age after a birthday.
- User1 changed the address after a move.

Running the example produced four tranlogs: tranlog2.dbf to tranlog5.dbf. The examples in this section are drawn from these tranlogs.

The `$GEMSTONE/examples/tranlogs` directory contains two files that contain the results of running `printlogs.sh` on the example tranlogs:

`demolog.txt` — Results of `printlogs.sh all`

`demologfull.txt` — Results of `printlogs.sh full all`

Tracking Changes to an Employee

Let's say we want to examine the change history of a particular Employee. Using the method #asOop, we find that the OOP of the Employee object of interest is 155053.

```
topaz 1> printit
| myEmployee |
myEmployee := <code to locate employee object>.
myEmployee asOop.
%
155053
```

The data composing an Employee is contained in subobjects (such as address), as well as directly (such as age). So, we will also need to track changes to these subobjects. Again using #asOop, we find that the OOP of the Name object associated with this Employee is 155073, and that the OOP for the Address object is 155069.

```
myEmployee name asOop
155073
myEmployee address asOop
155069
```

You can use #asOop on any persistent object in the repository. For example,

```
73 asOop
295
nil asOop
10
```

We can now search the tranlogs for any events involving these objects. Using the command:

```
$> searchlogs.sh 155053 155073 155069
```

results in the output shown in Example H.6.

Example H.6 searchlogs.sh Output for Employee

```
Searching for oops:  155053 155073 155069
Searching tranlogs:
  tranlog2.dbf
  tranlog3.dbf
  tranlog4.dbf
  tranlog5.dbf
```

```
3.61.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
  object 155053 cls 180689

3.61.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
  object 155069 cls 182393

3.61.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
  object 155073 cls 180101

3.61.1  Commit session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)

 crBacklog: 1  crPageId: 944
  beginLogRecord:(file:3 rec:61)  reclaimedOopsDebugInfo: 0
  timeWritten: [1156955988] Wed 10 Aug 2011 09:39:48 PDT

4.7.0  BeginData session: 2 user: User2 gemhost: merlin clientIP:
10.20.30.40 beginId:(142 2)
  object 155053 cls 180689

4.7.1  Commit session: 2 user: User2 gemhost: merlin clientIP:
10.20.30.40 beginId:(142 2)
 crBacklog: 1  crPageId: 872
  beginLogRecord:(file:4 rec:7)  reclaimedOopsDebugInfo: 0
  timeWritten: [1156956023] Wed 10 Aug 2011 09:40:23 PDT

5.7.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(151 2)
  object 155069 cls 182393

5.7.1  Commit session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(151 2)
 crBacklog: 1  crPageId: 875
  beginLogRecord:(file:5 rec:7)  reclaimedOopsDebugInfo: 0
  timeWritten: [1156956060] Wed 10 Aug 2011 09:41:00 PDT
```

From this output, we can see that in tranlog entry 3.61.0, User1 made changes to all three objects (in this case, when the Employee and associated subobjects were first created). In entry 4.7.0, User2 made a change to the Employee. Later, in entry 5.7.0, User1 made a change to 155069, the Address object.

Note that the `BeginData` entries are each followed by a `Commit`. You should always confirm that a `BeginData`/`Data`/`BeginStoreData`/`StoreData` entry is followed by a `Commit`. If it doesn't, then the reported event was not made persistent in the repository.

Changed vs. New Objects

In the above example, while the field of an Address object changed, the Address object itself was the same (had the same OOP). Depending on how the Smalltalk application is written, this may not always be the case. If application that was initiating these changes created a new Address object, and assigned the Employee's address instance variable to this new object, then the Employee object would reference a new OOP, rather than OOP 155069. This would make the analysis somewhat different. For example, in the initial stage of the analysis when you look up the OOP of the Address object in your application, you would find the new OOP rather than the original OOP. Looking back in time, you would see when this Address object was created and assigned to the Employee instance.

Details of Changes to an Employee

Having used the `searchlogs.sh` script to get a general idea of which tranlogs are of interest, you can now use the `printlogs.sh` script to get more details.

Let's say you want more details on the creation of Employee object 155069 and its associated subobjects 155073 and 155069 in entry 3.61.0. The "3" in "3.61.0" indicates that `tranlog3.dbf` is the tranlog of interest. The command:

```
$> printlogs.sh tranlog3.dbf
```

will generate a condensed listing of all events in `tranlog3.dbf`. By searching the resulting file for the entry number 3.61.0, you can find the relevant entry. See Example H.7.

Example H.7 Employee Modification

```
3.61.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
  clusterId: 1,  extentId: 0  numObjs:35

145021  155041  155045  155049  155053  155069  155073  155077
155089
155093  155097  155101  155121  155273  155285  155317  156589
156597  156609  156613  156661  156689  156729  156733  156749
156825  156829  156833  156837  156857  156861  156885  179629
180045  180653
```

This example shows the OOPs of *all* objects created during this event. If you want to see more details on the actual changes made, use the “full” command-line option when you run the `printlogs.sh` script:

```
$> printlogs.sh full tranlog3.dbf
```

This will produce a more detailed listing of all events. See Example H.8.

Example H.8 Employee Modification (Full Mode)

```
3.61.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
      clusterId: 1,  extentId: 0  numObjs:35
```

[details for other objects omitted]

```
objId: 155053 class: 180689 seg: 3253 sz: 3 bits: 0x47 psize: 32
Oop values:  155073  295  155069

objId: 155069 class: 182393 seg: 3253 sz: 5 bits: 0x47 psize: 40
Oop values:  156833  10  156837  156661  391335

objId: 155073 class: 180101 seg: 3253 sz: 3 bits: 0x47 psize: 32
Oop values:  156829  156825  10
```

[details for other objects omitted]

```
objId: 156661 class: 1169 seg: 3253 sz: 2 bits: 0x3007 psize: 24
Bytes:  OR^@^@
```

[details for other objects omitted]

```
objId: 156825 class: 1169 seg: 3253 sz: 7 bits: 0x3007 psize: 28
Bytes:  Patrick^@

objId: 156829 class: 1169 seg: 3253 sz: 5 bits: 0x3007 psize: 28
Bytes:  Ohara^@^@^@

objId: 156833 class: 1169 seg: 3253 sz: 13 bits: 0x3007 psize: 36
Bytes:  2556 Fir Blvd^@^@^@

objId: 156837 class: 1169 seg: 3253 sz: 7 bits: 0x3007 psize: 28
Bytes:  Ashford^@
```

[details for other objects omitted]

For the Employee object 155053, we find:

```
objId: 155053 class: 180689 seg: 3253 sz: 3 bits: 0x47
psize: 32
Oop values:  155073  295  155069
```

The tranlog tells us that the Employee is an instance of class 180689, the Employee class, which has three instance variables: name, age, and address. By position, we can identify the data in the instance variables:

name — 155073 (the OOP of an instance of Name)
age — 295 (the OOP of the SmallInteger 73)
address — 155069 (the OOP of a instance of Address)

Looking at the Name object (155073), we find:

```
objId: 155073 class: 180101 seg: 3253 sz: 3 bits: 0x47
psize: 32
Oop values:  156829  156825  10
```

This object is an instance of the class with OOP 180101 (Name). Name contains three instance variables: last, first, and middle. By position, we see the data is:

last — 156829 (the OOP of a String)
first — 156825 (the OOP of a String)
middle — 10 (the OOP of nil; in this example, no middle name was set)

For the last name object (156829), we find:

```
objId: 156829 class: 1169 seg: 3253 sz: 5 bits: 0x3007
psize: 28
Bytes:  Ohara^@^@^@
```

The last name is the string "Ohara". The ^@ indicate nulls in the String after its official end. There may be up to three nulls; Strings are adjusted to a size that is a multiple of four bytes.

By a similar process, you can examine the structure of other subobjects in the Name and Address objects.

Further Analysis

Class Operations

To find all objects created or modified that belong to a particular class, first generate `printlogs.sh full` output of the tranlogs of interest. Each time an object of that class is created or modified, the full tranlog entry includes the line

```
class: <OOP>
```

You can use the UNIX `grep` command to find all references to the OOP of the class.

For example, to find all creation or modification of any instance of the example class Employee:

1. Generate `printlogs.sh` full mode output (in this example, `demologfull.txt`).
2. Execute the `grep` command to find all instances of the Employee class (OOP 180689):

```
$>  grep "class: 180689" demologfull.txt
```

This produces output of the form:

```
objId: 155041 class: 180689 seg: 3253 sz: 3 bits: 0x47
psize: 32
objId: 155053 class: 180689 seg: 3253 sz: 3 bits: 0x47
psize: 32
objId: 155077 class: 180689 seg: 3253 sz: 3 bits: 0x47
psize: 32
objId: 155097 class: 180689 seg: 3253 sz: 3 bits: 0x47
psize: 32
objId: 155053 class: 180689 seg: 3253 sz: 3 bits: 0x47
psize: 32
```

This gives the first line from the entry creating/modifying the object belonging to that class. From this, you can use other commands to search for and/or track the history of these objects.

Deleted Objects

An object-oriented system doesn't actually delete objects; rather, objects cease to be referenced and are eventually garbage-collected. Noting the removal of an object requires you to examine the references to that object (such as from a collection) and identify when the referencing object was modified in such a way that the object of interest is no longer referenced. Meanwhile, as you examine the `printlogs.sh` output, you may find references to the OOP of a dereferenced object in garbage collection tranlog entries.

Managing Volume

As noted above, the `printlogs.sh` script produces a very large amount of output. GemStone tranlogs may be very large; while 2 GB is historically recommended, up to 16 GB are supported. The output of `printlogs.sh` in normal mode will be somewhat larger than the original tranlog (the `printlogs.sh` output, being human readable, is less dense). The output from this script in full mode is much larger.

To manage the volume, follow these guidelines:

- Avoid configuring your system with very large tranlogs.
- Before beginning analysis, ensure that you have plenty of disk space available.
- Print only the tranlogs containing data you need. Use the `searchlogs.sh` script to identify exactly where the required information is located.
- Make sure that only the relevant tranlogs are in the current directory; move the unneeded ones elsewhere. However, you must retain a continuous set of tranlogs without gaps in sequence, and you must include the tranlog with the original login entry, in order to have the UserId and other information provided.
- Once you have printed the output, use the UNIX utility `grep -n` to locate the lines of interest, and the UNIX utility `split -l` to break the resulting file up into chunks of a more manageable size.

Index