GemStone

# GemStone System Administration Guide

July 1996

## GemStone

Version 5.0
For UNIX

## IMPORTANT NOTICE

This manual and the information contained in it are furnished for informational use only and are subject to change without notice. GemStone Systems, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual or in the information contained in it. The manual, or any part of it, may not be reproduced, displayed, photocopied, transmitted or otherwise copied in any form or by any means now known or later developed, such as electronic, optical or mechanical means, without written authorization from GemStone Systems, Inc. Any unauthorized copying may be a violation of law.

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

Copyright 1996 by GemStone Systems, Inc. All rights reserved.

Use, duplication, or disclosure by the Government is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

### Trademarks

**GemStone** is a registered trademark of GemStone Systems, Inc.

**Kerberos** is Copyright (C) 1989 by the Massachusetts Institute of Technology. Export of this software from the United States of America is assumed to require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting. Within that constraint, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty.

**Sun**, **Sun Microsystems**, **Solaris** and **SunOS** are trademarks or registered trademarks of Sun Microsystems, Inc. All **SPARC** trademarks, including **SPARCstation**, are trademarks or registered trademarks of SPARC International, Inc. **SPARCstation** is licensed exclusively to Sun Microsystems, Inc.

**UNIX** is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

*Preface*

# About This Manual

This manual tells how to perform day-to-day administration as the GemStone® "data curator."

Installation instructions are included with your GemStone Installation Guide, which should be kept with this manual.

This manual is organized in three parts, configuration, administration, and concepts, which are described in the following paragraphs.

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. Four sample configuration files are provided as starting points.
- Chapter 2, "Configuring Gem Session Processes," tells how to configure the GemStone processes that provided 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 start-up problems, and how to deal with unexpected shutdowns.
- Chapter 5, “Launching the Administration Tools,” previews the tools available for administration tasks and how to log in to the repository.
- Chapter 6, “Administering User Accounts and Security,” explains how to create, modify, and remove GemStone user accounts. It also tells how to set up and modify a user’s privileges and authorizations.
- Chapter 7, “Managing the Repository,” gives procedures for managing the repository itself: checking free space, adding space, controlling its growth, and auditing it for consistency. It also how to recover from file system corruption.
- Chapter 8, “Managing Transaction Logs,” gives procedures for setting up the optional full incremental logging, managing log space, and archiving the log files.
- Chapter 9, “Making and Restoring Backups,” gives procedures for making a GemStone full backup while the repository is in use, and for restoring the repository from the full backup and the optional full transaction logs.
- Chapter 10, “Monitoring GemStone,” explains what you should do to monitor the system in a way that doesn’t interfere with reclaim activity (garbage collection), where the system logs are located, and what additional performance monitoring methods are provided.

Part 3: Concepts for the System Administrator

- Chapter 11, “GemStone Garbage Collection,” provides background information about the implementation of the garbage collection function in GemStone.

There are eight appendices:

- 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 that are defined for use by the GemStone data curator.
- 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, “GemStone Directory Contents,” lists the contents of the GemStone system and data directories that are created when the GemStone software is installed.
- Appendix F, “Environment Variables,” lists all environment variables used by GemStone, including those that are reserved.
- Appendix G, “Translation Files for Messages,” explains the syntax and semantics of the GemStone language-dependent file for messages and how you can create a similar file in another language.
- Appendix H, “Kerberos,” explains how to configure the Kerberos authentication system for use with GemStone and, if you do not have the MIT Project Athena distribution, how to install Kerberos from executables in the GemStone distribution.

## Typographical Conventions

This document uses the following typographical conventions:

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

  **copydbf**

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

  ```
  markForCollection
  ```

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

  ```
  successful login
  ```

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

  ```
  topaz> set gemstone myStone
  ```

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

  *StoneName*`.conf`

In formal syntax listings, these additional conventions are used:

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

  **tcp**

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

  [*dbfName*]

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

  {*modifier*}

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

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

  *gemStoneName* | *netLdiName*

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

# Technical Support

GemStone provides several sources for product information and support. GemStone product manuals provide extensive documentation, and should always be your first source of information. GemStone Technical Support engineers will refer you to these documents when applicable. However, you may need to contact Technical Support for the following reasons:

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

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

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

- Your name, company name, and GemStone license number,
- the GemStone product and version you are using,
- the hardware platform and operating system you are using,
- a description of the problem or request,
- exact error message(s) received, if any.

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

For non-emergency requests, you should contact Technical Support by email, Web form, or facsimile. You will receive confirmation of your request, and a request assignment number for tracking. Replies will be sent by email whenever possible, regardless of how they were received.

**Email:** **support@gemstone.com**
The preferred method of contact. Please do not send files larger than 100K (for example, core dumps) to this address. A special address for large files will be provided on request.

**World Wide Web**: **http://www.gemstone.com**

Technical Support is located under Services. A Help Request Form is available for request submissions. This form requires a password, which is free of charge but must be requested by completing the Registration Form, found in the same location. Allow 24 hours for your registration to be recorded and a password assigned.

**Facsimile**: **(503) 629-8556**

When you send a fax to Technical Support, you should also leave a voicemail message to make sure your fax will be picked up as soon as possible.

We recommend you use telephone contact only for more serious requests that require immediate evaluation, such as a production database that is non-operational.

**Telephone**: **(800) 243-4772 or (503) 690-3503**

Emergency requests will be handled by the first available engineer. If you are reporting an emergency and you receive a recorded message, do not use the voicemail option. Transfer your call to the operator, who will take a message and immediately contact an engineer.

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

## 24x7 Emergency Technical Support

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

Contents

# Part I: System Configuration

## Chapter 1. Configuring the GemStone Server

## *Chapter 2. Configuring Gem Session Processes*

## *Chapter 3. Connecting Distributed Systems*

# Part II: System Administration

## Chapter 4. Running GemStone

## *Chapter 5. Launching the Administration Tools*



## *Chapter 6. Administering User Accounts and Security*

## *Chapter 7. Managing the Repository*

## *Chapter 8. Managing Transaction Logs*

## *Chapter 9. Making and Restoring Backups*

## Chapter 10. Monitoring GemStone



# Part III: Concepts for the System Administrator

## Chapter 11. GemStone Garbage Collection

# *Appendices*

## *Appendix A. GemStone Configuration Options*

## *Appendix B. GemStone Utility Commands*



## *Appendix C. Network Resource String Syntax*

## *Appendix D. GemStone Kernel Objects*



## *Appendix E. GemStone Directory Contents*



## *Appendix F. Environment Variables*

## *Appendix G. Translation Files for Messages*



## *Appendix H. Kerberos*

# List of Figures

# List of Tables

GemStone

# Part I: System Configuration

Chapter 1

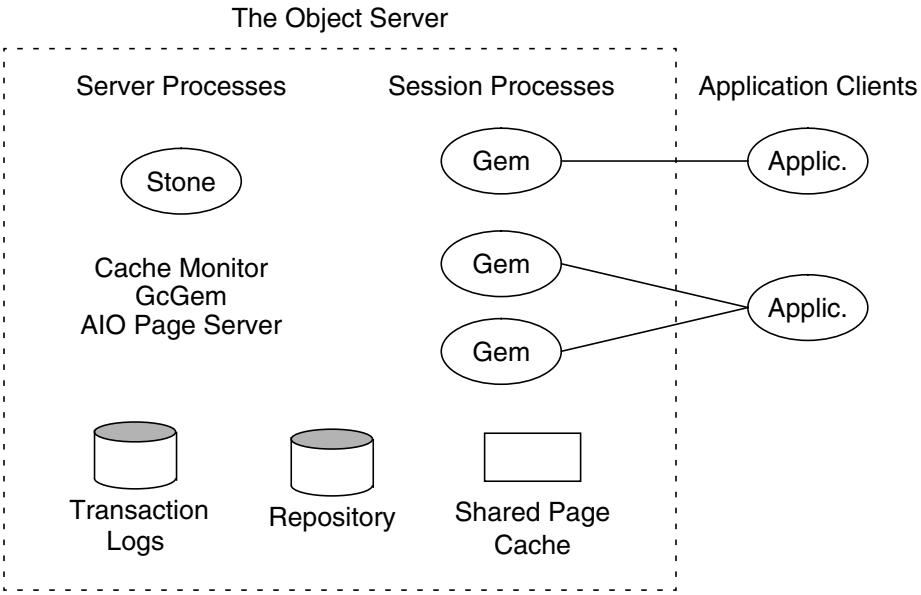
# Configuring the GemStone Server

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 three 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 1.2 shows the key parts that define the server configuration:

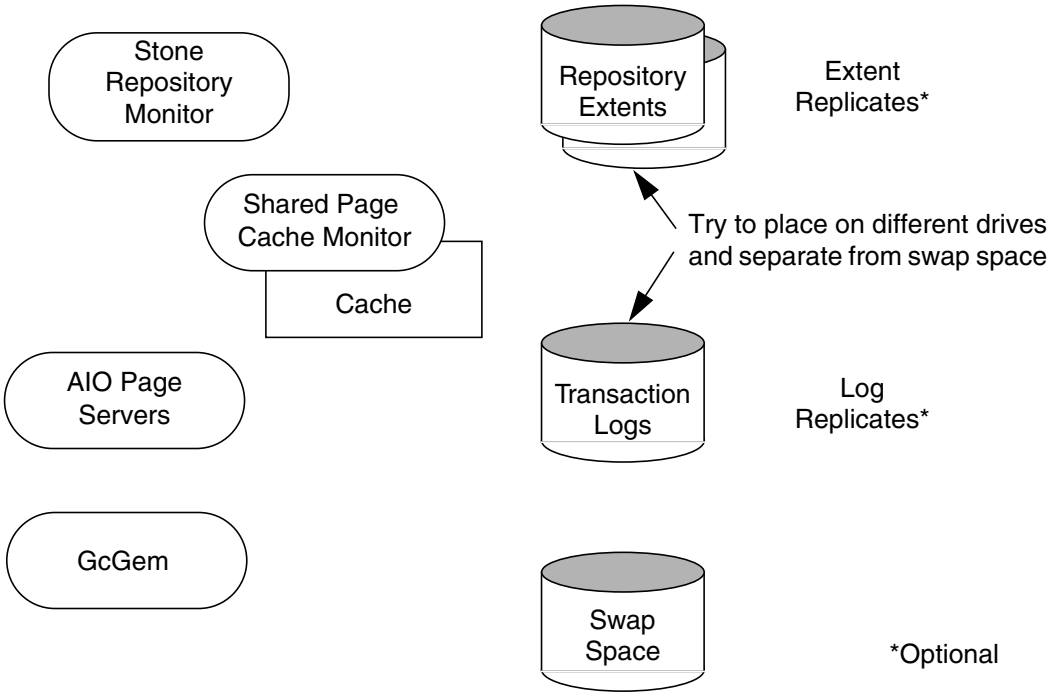
- The *Stone* repository monitor process acts as a resource coordinator. It synchronizes critical repository activities and ensures repository consistency.
- The *shared page cache monitor* creates and maintains a shared page cache for the GemStone server. The monitor balances page allocation among processes, 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.
- 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 *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 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).

**Figure 1.2 Players in the GemStone Configuration Cast**



## 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 removing the # symbol.
- 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 MBytes.

*NOTE*

*All modifications that define a particular sample configuration are grouped at the end of that file.These modifications override the initial settings, which occur earlier in the file. The location was chosen so you could easily identify the changes.*

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 increased throughput. It accommodates more uses 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 Gbytes. 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 500 users and a repository of 30 Gbytes. 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.)

If you want more information about any of these settings, see the detailed instructions for establishing your own configuration beginning on page 1-10.

**Table 1.1 Configuration Settings for Selected Repository Sizes and Number of Users**

| Characteristic or Configuration Option | Server Configuration | | | |
|---|---|---|---|---|
| | **Small** | **Medium** | **Large** | **Very Large** |
| **Application Characteristics** | | | | |
| Maximum number of user sessions | 12 | 50 | 200 | 500 |
| Size of repository (GBytes) | 0.100 | 1 | 10 | 30 |
| **System Requirements** | | | | |
| Typical Number of CPUs | 1 | 1 | 2–4[a] | 4 or more[a] |
| Total real memory (MBytes) | 64[b] | 256 | 1000 | 2000 |
| Kernel shared memory (MBytes) | 23 | 101 | 502 | 1006 |
| Number of disk drives | 3[c] | 3[c] | 7[c] | 10[c] |
| **Configuration Settings** | | | | |
| STN_MAX_SESSIONS | 40[d] | 50 | 200 | 500 |
| STN_SIGNAL_ABORT_CR_BACKLOG | 20 | 75 | 300 | 700 |
| STN_PRIVATE_PAGE_CACHE_KB | 1600 | 3500 | 10000 | 25000 |
| STN_NUM_LOCAL_AIO_SERVERS | 1 | 1 | 1 | 2 |
| SHR_PAGE_CACHE_SIZE_KB | 21000 | 100000 | 500000 | 1000000 |
| **Approximate Memory Usage** | | | | |
| Stone repository monitor (MBytes) | 3 | 6 | 20 | 40 |
| Each Gem session process (MBytes) | 2.5 | 2.5 | 2.2 | 2.0 |

[a] These numbers are based on commercially available systems at the time of this release.

[b] The total memory should be 128 Mbytes under Solaris 2.5 and under UNIX systems with no limit on the size of file system buffers. On systems that permit it, the file system buffers should be limited to about 10% of real memory.

[c] 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.

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

Table 1.2 gives changes recommended for efficient operation and protection against system faults. Some of these, such as the use of raw disk partitions, also depend on the size of the application.

**Table 1.2 Configuration Settings Recommended for Efficiency and Fault Protection**

| Configuration Option | Server Configuration | | | |
|---|---|---|---|---|
| | **Small** | **Medium** | **Large** | **Very Large** |
| **Extents** | | | | |
| DBF_EXTENT_NAMES | (1 file) | (1 raw partition) | (5 raw partitions) | (8 raw partitions) |
| DBF_EXTENT_SIZES | (unlimited) | 999[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 |
| DBF_REPLICATE_NAMES | (recommended) | (recommended) | (recommended) | (recommended) |
| **Transaction Logs** | | | | |
| STN_TRAN_FULL_LOGGING | True | True | True | True |
| STN_TRAN_LOG_DIRECTORIES | (2 directories) | (2 or more partitions) | (4 partitions) | (4 partitions) |
| STN_TRAN_LOG_SIZES | 10, 10 | 199[a] each | 499[a] each | 499[a] each |
| STN_REPL_TRAN_LOG_ DIRECTORIES | (recommended) | (recommended) | (recommended) | (recommended) |
| **Stone Response to Gem Fatal Errors** | | | | |
| STN_HALT_ON_FATAL_ERROR | True[b] | False | False | False |

[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 GByte partitions in the Large configuration and 4 GByte partitions in the Very Large configuration.

[b] For deployed systems, a setting of False is recommended.

# Recommendations About Disk Usage

You can enhance server performance by distributing the repository files on multiple disk drives and placing the data in raw disk partitions rather than in a file system.

## 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 (in a raw partition), 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 two guidelines that you should consider in their listed priority when developing your own configuration:

1. Keep extents and transaction logs separate from UNIX 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.

*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.*

## Why Use Raw Partitions?

Placing extents and transaction logs on raw disk partitions (sometimes called raw devices or raw logical devices) ordinarily yields better performance while reducing the memory overhead for file system buffers. Each raw partition is like a single large sequential file, with one extent or one transaction log per partition.

The use of raw partitions 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.

Although raw partitions are recommended, query-intensive applications may achieve adequate performance using extents in the file system by restricting the size of the file system buffers. If you don't limit the buffer size, the buffers may occupy so much memory that GemStone processes are swapped out excessively. See your operating system documentation.

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

## Developing a Replication Strategy

The use of replicated extents is recommended in Table 1.1 as a means of increasing system tolerance to media failure. 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.

There are two needs to consider:

- Applications that cannot tolerate loss of committed transactions should replicate the transaction logs and use full transaction logging. A replicated 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 loss of an extent.
- Applications that require rapid recovery from loss of an extent (that is, without the delay of restoring from a backup) should replicate all extents on other devices in addition to replicating transaction logs. Changes to DBF_EXTENT_NAMES and DBF_REPLICATE_NAMES in the configuration file are all that is necessary before restarting GemStone. Restoring large repositories (many GBytes) from a backup may take several hours.

Hardware replication may provide a suitable alternative to GemStone replicates if the following points are kept in mind while designing the system:

- Extents benefit from efficiency of both random access (8 KByte repository pages) and sequential access (up to 128 MBytes 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.

# 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 UNIX and other software.

- The Stone and related processes need between 4 and 40 MBytes 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 MBytes to one GByte.

  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 two MBytes of memory on the node where it runs (see the discussion of memory needs for session processes on page 2-4). Each Gem process that runs on a remote (client) node also needs about 0.2 MBytes 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 Mbytes of RAM should have at least 512 Mbytes 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 UNIX and other software such as the X Window System.

## Estimating File Descriptor Needs

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

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

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

### Checking the System Clock

The system clock should be set to the correct time. When GemStone opens the repository at startup, it compares the current system time with the recorded checkpoint times as part of a consistency check. A system time earlier than the time at which the last checkpoint was written may be taken as an indication of corrupted data and prevent GemStone from starting. The time comparisons use GMT. It is not necessary to adjust GemStone for changes to and from daylight savings time in the United States.

## To Set the Page Cache Options and the Number of Sessions

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

### Shared Page Cache

The GemStone shared page cache system consists of two components, the shared page cache itself and a monitor process (`shrpcmonitor`). Figure 1.3 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.

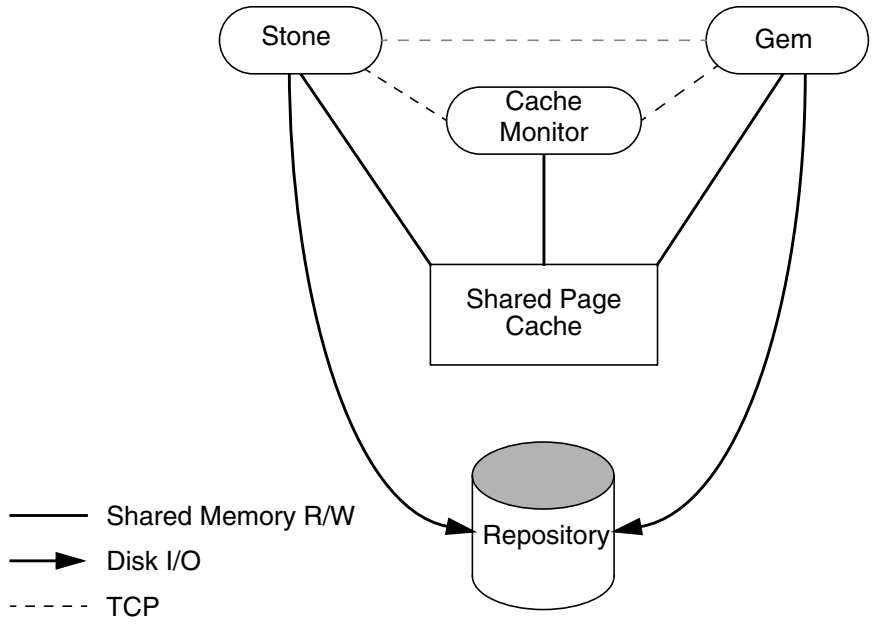
The shared page cache resides in a segment of the operating system's virtual memory that is available to any authorized process. When the Stone repository

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, `gemserver50192.83.233.25.` 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 1.3 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 MBytes):

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 (2Bytes * 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 MBytes of the repository to the previous total,
   add 6% of the next 900 MBytes, and
   add 3% of that portion greater than 1 GByte.
3. the number of users and the size of their transactions—we recommend adding 0.5 to 1 MByte per user for most situations:

   add 1 MByte per user for the first 10 users,
   add 0.5 MByte 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 MBytes * 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 "To Monitor Page Reads and Writes by a Session" on page 10-7, especially the statistic NumberOfFreeFrames.

## 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 MByte is sufficient for up to 5 sessions. Increase that setting by 1 MByte 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 or your own estimate derived above (remember to convert to KBytes). 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 100 MByte cache,

```
SHR_PAGE_CACHE_SIZE_KB = 100000;
```

**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 KBytes per user. Add 40 KBytes per user for sessions that don't acquire locks, and 64 KBytes 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 KBytes 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 + 3).

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 KByte of cache space plus about 20 KBytes 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; for example,

```
% $GEMSTONE/bin/bash
bash$ ulimit -a
```

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.

## 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 MByte 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 MByte), the Stone takes a number of steps to avoid shutting down. For information, see "How to Recover from Disk-Full Conditions" on page 4-22.

**Example 1.1 Extent Size Including Working Space**

```
Size of repository                 = 1 GByte

Free-Space Allowance
  .5 MByte * 20 sessions           = 10 MBytes

Fragmentation Allowance
  1 GByte * 15%                    = 150 MBytes

Total with Working Space           = 1.16 GBytes
```

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 up-

grading 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.
- Storing raw data in a disk partition (rather than in a file system) can yield somewhat better performance. For query-intensive applications, an alternative available under some operating systems is to use the file system, but limit its buffer size.
- If GemStone will be used in a distributed installation, it's best if all (or at least most) extents reside on the node where the Stone repository monitor is running. At least one extent or replicated extent *must* reside on the Stone's node. All Gem session processes must have access to the extent files, either by running on the same node or by a network connection. Placing extents on a remote node requires that a NetLDI (Network Long Distance Information) server be running on that node. (For further information, see "To Set Up Remote Extents or Replicates" on page 3-33.)

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 MBytes 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.

*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, set the size to 1995 MBytes for a 2 GByte partition.*

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 MBytes each in raw partitions.

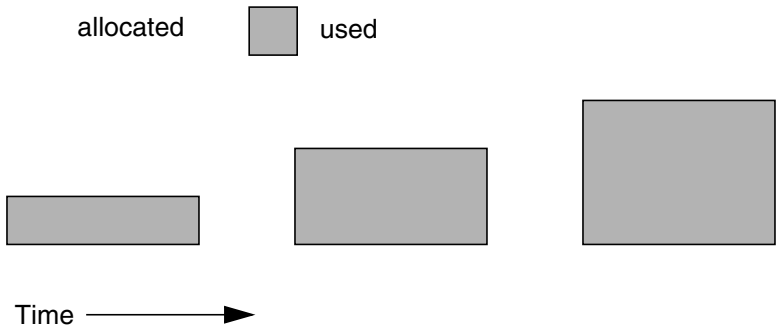
```
DBF_EXTENT_NAMES = /dev/rdsk/c1t3d0s5, /dev/rdsk/c2t2d0s6;
DBF_EXTENT_SIZES = 498, 498;
```

The maximum size of an extent is limited by the operating system. For information about system dependencies, see the maximums given under "DBF_EXTENT_SIZES" on page A-11.

## Pre-growing 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.

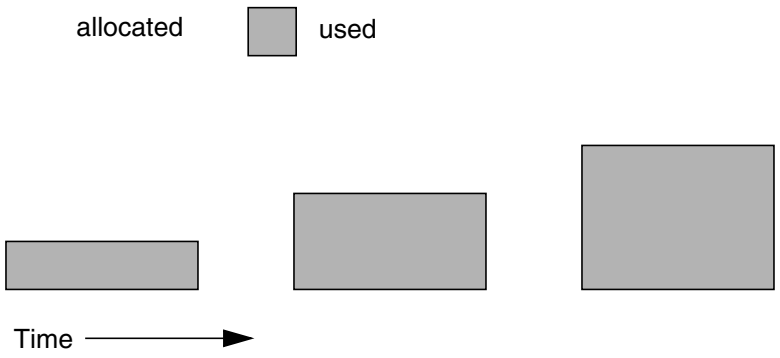
**Figure 1.4 Growing an Extent on Demand**




You can increase I/O efficiency while reducing file system fragmentation by instructing GemStone to allocate an extent to a predetermined size (called pre-growing 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.

Pre-growing 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. Pre-growing extents also provides a simple way to reserve space on a disk for a GemStone extent.

**Figure 1.5 Pre-Growing an Extent**




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

Two configuration options work together to pre-grow 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 pre-grow 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.

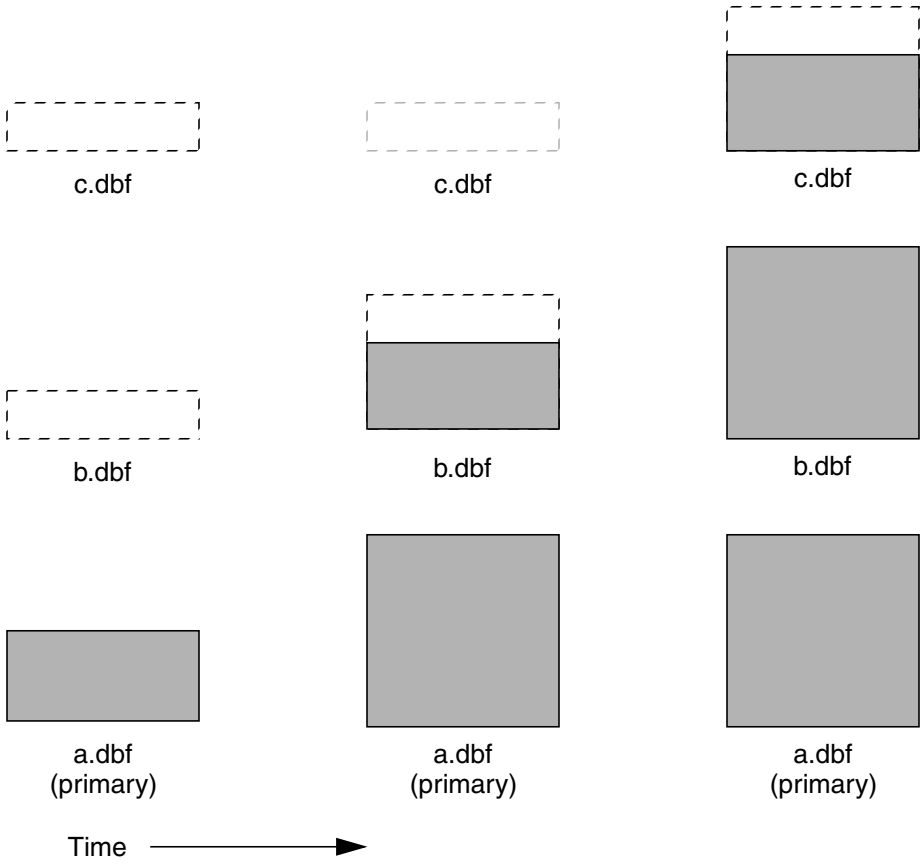
The DBF_ALLOCATION_MODE configuration option determines whether GemStone allocates new disk pages to multiple extents by filling each extent sequentially or by balancing the extents using a set of weights you specify. If you have placed

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.

**Figure 1.6 Sequential Allocation**



## 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;
```

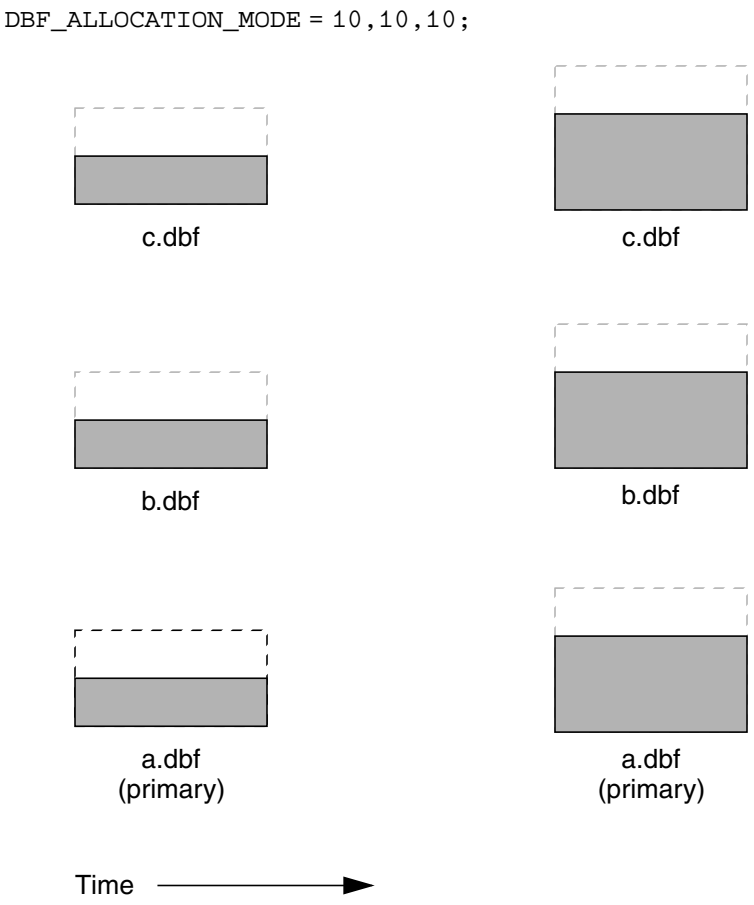
You can think of the total weight of a repository as the sum of the weights of its extents. When the Stone allocates space from the repository, each extent contributes an allocation proportional to its weight.

*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.*

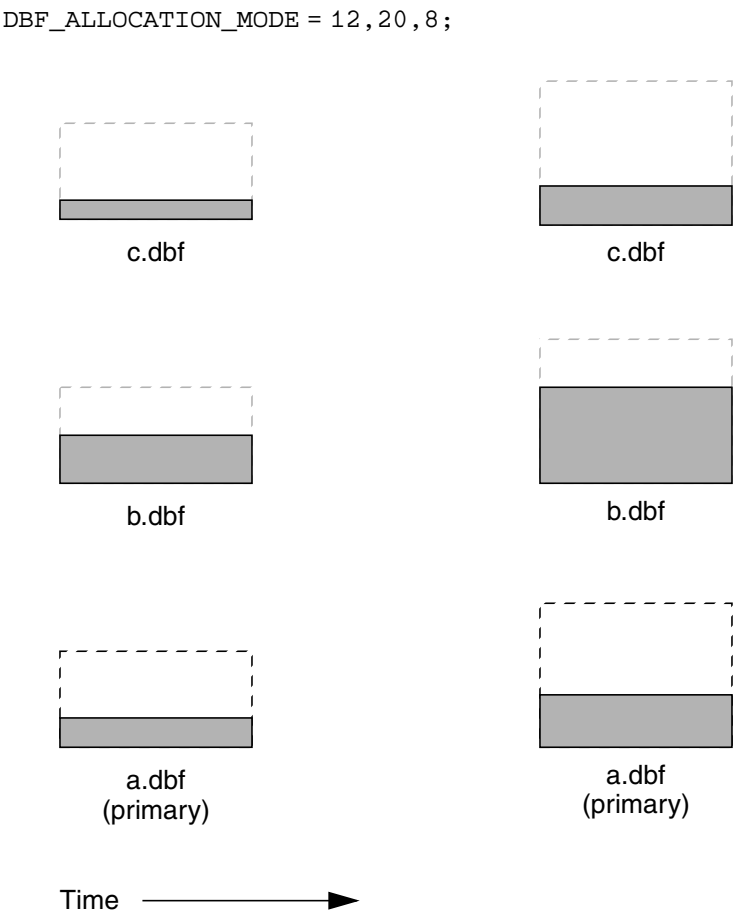
One reason for specifying weighted allocation of a repository's extents is to share the I/O load among the extents. For example, you can create three extents with equal weights, as shown in Figure 1.7.

**Figure 1.7 Equally-Weighted Allocation**




Although equal weights are most common, you can adjust the relative extent weights for other reasons, such as to favor a faster disk drive. For example, suppose we have defined three extents: A, B, and C. If we defined their weights to be 12, 20, and 8 respectively, then for every 40 disk units (pages) allocated, 12 would come from A, 20 from B, and 8 from C. Another way of stating this formula is that because B's weight is 50% of the total repository weight, 50% of all newly-allocated pages are taken from extent B. Figure 1.8 shows the result.

**Figure 1.8 Proportionally-Weighted Allocation**




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.*

### Effect of Clustering on Allocation Mode

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

### Weighted Allocation for Extents Created at Run Time

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

## 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.

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 removed 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 7-7. You may also want to create replicates of transaction logs; see “To Add a Log and Replicate at Run Time” on page 8-9.

# 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 8-11.

For general information about the logging mode and the administrative differences, see "Logging modes" on page 8-2.

## 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 on line 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 KBytes

Archiving interval                                = Daily

Transactions between Archives
  5 per minute * 60 minutes * 15 hours            = 4500

Log space (minimum)
  4500 transactions * 5 KBytes                    = 22 MBytes
```

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 MBytes 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.

*WARNING*

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

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

Although the default size of 10 MBytes is adequate in some situations, update-intensive applications should consider a larger size (at least 25 to 50 MBytes) to limit the frequency with which logs are switched. Each switch causes a checkpoint to occur, which can impact performance.

*NOTE*

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

The following example sets up a log in a 2 Gbyte raw partition and a directory of 50 Mbyte 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;
```

Although any or all of the transaction logs can reside on a node remote from the Stone, for optimal performance they should be on the Stone's node. Placing logs on a remote node requires that a NetLDI be running on that node. (For further information, see "To Set Up Remote Transaction Logs or Log Replicates" on page 3-36.)

## 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 log replicates, 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 8-9.

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_ERROR = FALSE;
```

## To Set File Permissions for the Server

The primary consideration in setting file permissions for the Server is to protect the repository extents. All reads and writes should be done through GemStone repository executables: the Stone and its child processes (the shared page cache monitor, AIO page server, and GcGem) 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 | -rw------- | gsadmin | gsgroup | | Users read and write repository through GemStone processes. The Stone sets up the page cache in shared memory. |
| shared memory | -rw-rw---- | gsadmin | gsgroup | | |
| stoned | -r-sr-xr-x | gsadmin | gsgroup | gsadmin | AIO page server and GC Gem are spawned by stoned and can access repository as the gsadmin account. |
| pgsvrmain | -r-sr-xr-x | gsadmin | gsgroup | gsadmin | |
| gem | -r-sr-xr-x | gsadmin | gsgroup | gsadmin | |

[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 2-7.

## 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.

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

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

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

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

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

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

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

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

# Changing Between Files and Raw Partitions

This section tells you how to change your configuration by moving existing repository extent files to raw partitions or by moving existing extents in raw partitions to files in a file system. 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 MBytes 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 MBytes 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 MBytes 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 MBytes 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 2-9):

`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> level 3
topaz 1> printit
System stoneConfigurationReport
%
a SymbolDictionary
...
#1 a SymbolAssociation
            key       STN_REMOTE_CACHE_TIMEOUT
            value     5
#2 a SymbolAssociation
            key       DBF_ALLOCATION_MODE
            value     SEQUENTIAL
#3 a SymbolAssociation
            key       SpinLockCount
            value     1
...
```

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 | 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:` *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_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. Since that is a negative SmallInteger, 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 *GemStone Kernel Reference*.

# 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

Three 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 1-15) in light of your application’s design and usage patterns. Estimates of the number of objects queried or updated are particularly useful in tuning the cache.

You can use the shared page cache statistics for a running application to monitor the amount of unused space. See “To Monitor Page Reads and Writes by a Session” on page 10-7, especially the statistic FreeFrameCount.

### Locking the Cache in Memory

If your operating system permits it, locking down the cache improves performance by preventing swapping. On some systems, such locking is required for asynchronous I/O, which also improves performance. When the SHR_PAGE_CACHE_LOCKED configuration option is set to true, GemStone attempts to lock the cache. However, the operating system may restrict this action to processes running as root or may require special privileges (such as PRIV_MLOCK on HP-UX) for the option to take effect. For further information, consult your operating system documentation.

### 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 MBytes.
- Increase the size of the shared page cache. The default backlog of 20 commit records requires about one MByte, assuming a typical small transaction occupies about 50 KBytes.

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 | 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.
- 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 10-15.

A checkpoint also occurs each time the Stone repository monitor is shut down gracefully, as by invoking **stopstone** or `System | 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.

## 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 pre-grow multiple extents.)

If your configuration has two or more 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. Values greater than four are unlikely to have any benefit. For further information about this configuration parameter, see page A-23.

# 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 v5.0 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 1-8.

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 *devserver50*. GemStone will create log files with that server name as the prefix.

```
$ startstone devserver50
```

**Step 5.** Start linked Topaz, then set the GemStone name to *devserver50* and log in as DataCurator:

```
% topaz -l
topaz> set gemstone devserver50
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

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

For general information about restoring backups and transaction logs, see "How to Restore a GemStone Repository" on page 9-7. This discussion assumes you are familiar with that procedure.

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

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

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

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

**Step 4.** Decide on a naming convention or location that you will use on the duplicate server to keep the archive logs separate from those being created by the duplicate server itself. For instance, if both Stones use the default prefix of *tranlog*, you might copy `tranlog123.dbf` on the production server to $GEMSTONE/data/`prodtranlog123.dbf` on the duplicate server.

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

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

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

following configuration parameters for the *production* Stone: STN_TRAN_LOG_DIRECTORIES, STN_TRAN_LOG_PREFIX, STN_REPL_TRAN_LOG_DIRECTORIES, and STN_REPL_TRAN_LOG_PREFIX. For details, see the method description in the *GemStone Kernel Reference*.

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

The following example uses the names from Step 4:

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

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

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

**Step 8.** Repeat Steps 6 and 7 as necessary.

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

Chapter 2

# Configuring Gem Session Processes

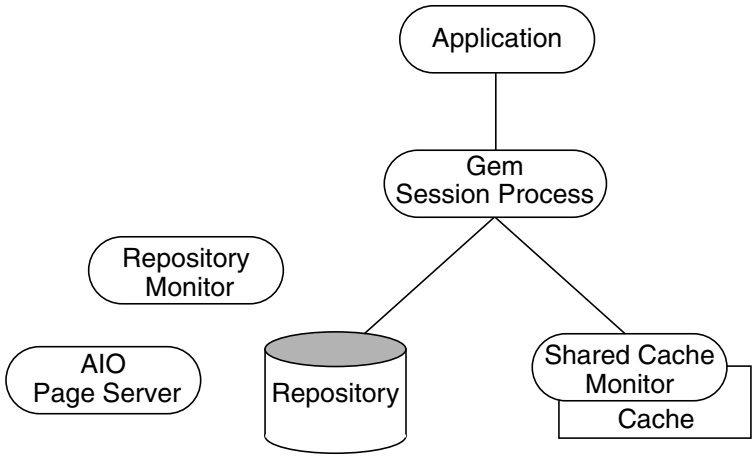
This chapter tells how to configure the GemStone session processes for your application. For additional information about running session processes on a node remote from the Stone repository monitor, refer also to Chapter 3.

## 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. If you have a choice, in general the linked version gives better performance because it involves less overhead. However, C and 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 MBytes of memory, of which 1.5 MBytes is for code that can be shared by other session processes. Each additional session process requires about 2.5 MBytes. The requirement is the same for Gems linked with an application. If you tune the cache size for Gems (page 2-11), add any increase to the amount given here. 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 1-11.

## Estimating Swap Space Needs

Swap (paging) space on machines remote from the Stones should follow the same general guidelines given on page 1-11 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.

## Set the Gem Configuration Options

The configuration options you should focus on initially are the ones for the shared page cache. Changes to other Gem configuration options are discussed later under "How to Tune Session Performance" on page 2-11.

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

On other nodes, the first Gem session process to log in ordinarily causes the Stone to start a shared page cache monitor and cache on that node. Parameters in the Gem's configuration file determine the size of the cache and the number of processes that can attach to it (SHR_PAGE_CACHE_SIZE_KB and SHR_PAGE_CACHE_NUM_PROCS, respectively). All subsequent sessions that log in from that remote node will be directed to use the same cache.

Although it is possible to disable the shared page cache on a node remote from the Stone and extents, that option is intended for use by session processes on certain platforms that do not support shared memory or where there is only one Gem on the node. You should leave the shared page cache enabled whenever more than one Gem session process will be logged in to the same repository from the same node. The shared page cache is enabled by the following line, which shows the default setting:

```
GEM_SHR_PAGE_CACHE_ENABLED = TRUE;
```

If you disable the shared page cache on a node remote from the Stone, be sure that the size of the private page cache for each Gem session process is sufficient to compensate. The GEM_PRIVATE_PAGE_CACHE_KB configuration option has a second size parameter that is used in the absence of a shared cache. See "To Tune the Private Page Cache" on page 2-11.

You can use the GemStone **shmem** utility described on page 1-16 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 1-31. 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 a shared page cache, the UNIX user account of the linked application or Gem session process must either be the same as that of the Stone (such as the *gsadmin* account) or be one that belongs to the same group as the Stone. The same requirement applies to page server processes, which are discussed in Chapter 3, "Connecting Distributed Systems."

  If the setuid bit is set on repository executables as recommended in Table 1.3 on page 1-32, 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 1-31) 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 1-31. This works well for **topaz -l**. The `installgs` script offers to set the file ownership and permissions for you. To do it manually, do this while logged in as root:

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

You may prefer not to use the setuid bit with linked applications that do not distinguish between real and effective user IDs. GemStone's Topaz executable performs repository reads and writes as the *effective* user (the account that owns the executable's file), but performs other reads and writes as the *real* user (the one who invoked it). Linked applications that do not make this distinction, such as a third-party Smalltalk used with GemBuilder, are likely to perform *all* I/O as the effective user, or *gsadmin*. If this result is unsatisfactory, remove the S bit on that executable and add group write permission to the extents.

## To Set Access for All Other Applications

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

## To Set Access to Other Files

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

| | |
|---|---|
| $HOME | GemStone ordinarily creates log files for spawned processes (such as RPC Gem session processes and page servers) in the home directory of the user or the NetLDI captive account. In situations where the home directory cannot be writable, the environment variable GEMSTONE_DEFAULT_NRS can be used to specify an alternative location; see "To Set a Default NRS" on page 3-15. |
| `/opt/gemstone` | All users should have read/write/execute access to the directories `/opt/gemstone/log` and `/opt/gemstone/locks` on each host. (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 | 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_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 these methods, see the *GemStone Kernel Reference*. 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 KBytes and 3 MBytes. Large applications typically require a temporary object space of 1 to 1.5 MBytes. You may find it helpful to examine the cache statistics NotConnectedObjsSetSize and MakeRoomInOldSpaceCount; see "To Monitor Cache Statistics" on page 10-8.

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

It is seldom necessary to change the GEM_PRIVATE_PAGE_CACHE_KB setting unless the Gem is performing a special operation, such as a garbage collection scan of the repository (`markForCollection`) or bulk loading of the repository.

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 KBytes) 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.

The GEM_PRIVATE_PAGE_CACHE_KB configuration option takes two values, one for use when a shared page cache exists on the Gem's node, and the other for when a shared cache does not exist. If the shared page cache is disabled on a node remote from the Stone, the Gem's private page cache also is used when reading pages from the disk. In that case, the GEM_PRIVATE_PAGE_CACHE_KB configuration option should be increased to compensate. A minimum of 6000 KBytes is recommended. For example, using the default sizes:

```
GEM_PRIVATE_PAGE_CACHE_KB 200, 6000;
```

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 Tune Garbage Collection of the NotConnectedSet

Each Gem session process keeps track of new objects it has moved to a repository page but which are not referenced by permanent objects. Some objects in this *notConnectedSet* may be temporary objects destined to become garbage; others eventually become committed, permanent objects. Initially, all of them are referenced only from that session's memory space. The notConnectedSet protects these objects from garbage collection until their fate is known. However, if the pages containing these objects are written to the disk and are committed, the objects that turn out to be garbage must be garbage collected later using more costly techniques. The goal is to prevent unnecessary growth of the repository by detecting and reclaiming these objects as soon as possible.

Each time the Gem performs a garbage collection of its memory space, it also examines the total number objects in its notConnectedSet. If the size of this set meets parameters given below, the Gem performs a garbage collection analysis of the objects. Any objects that are no longer referenced from the Gem's memory

space or the committed repository are garbage collected by the Gem or made available for the next epoch garbage collection of the repository.

## Parameters

Two internal parameters control garbage collection of a Gem's notConnectedSet:

`#NotConnectedThreshold`
: the minimum number of notConnected objects to garbage collect; the default is 2000 objects

`#NotConnectedDelta`
: the change in size of the notConnectedSet that triggers another garbage collection; the default is 300 objects

At the end of each in-memory garbage collection, the Gem performs a garbage collection analysis on its notConnectedSet if the size of the set is greater than `#NotConnectedThreshold` and the change in size since the last garbage collection is greater than `#NotConnectedDelta` .

For information about modifying these or other internal parameters, see "To Change Settings at Run Time" on page 2-9.

## Statistics

The method `System cacheStatistics:` for a Gem's cache slot reports three statistics that are useful for monitoring collection of the notConnected set:

`NotConnectedObjsSetSize`
: the number of objects in repository pages (in memory or on the disk) that are not connected to one of the permanent root objects in the repository

`GcNotConnectedCount`
: the number of times the notConnected objects have been garbage collected since the session started

`GcNotConnectedDeadCount`
: the number of dead objects found during garbage collection of the notConnected objects

For information about obtain shared page cache statistics for a Gem session process, see "To Monitor Cache Statistics" on page 10-8.

### Performance Factors

Garbage collection of the notConnectedSet consumes both CPU and I/O resources from the particular Gem session process. Tuning this garbage collection may involve tuning both the collection process itself and the part of your application that creates the temporary objects.

Monitor the three cache statistics described above and also `TimeInScavenges`, which is the CPU time in milliseconds spent by the Gem's in-memory garbage collector. Use these statistics to determine how often the notConnected garbage collector is running, how much CPU time it is consuming, and the amount of garbage it is finding. Based on the results, you may want to change the frequency with which the notConnectedSet is garbage collected by doing one or more of the following:

- changing the setting of the GEM_TEMPOBJ_CACHE_SIZE configuration option (a larger value means that the notConnectedSet grows more slowly),
- changing the `#NotConnectedThreshold` and `#NotConnectedDelta` parameters, or
- as a quick fix, having the application session log out (which discards all temporary objects) and then log in again.

## 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 A-4.)

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 2-9.

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 Primitive

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 primitive, such as those invoked by `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 | 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" on page 10-22.

### 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 1-42.

## 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/bin/services.dat` . A NetLDI checks that file to translate the name of a service to a command it can execute. For example:

```
MyGemnetObject $GEMSTONE/sys/MyGemnetObject
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 6-11.

Chapter 3

# Connecting Distributed Systems

This chapter tells how to set up GemStone in a distributed environment. There are four main topics following the overview:

- *How to Arrange Network Security* describes three ways to provide access to GemStone processes on other nodes.
- *How to Use Network Resource Strings* tells how to specify where distributed GemStone resources are located.
- *How to Set Up a Remote Session* gives step-by-step examples for setting up typical distributed client-server configurations. It also contains troubleshooting tips.
- *How to Set Up Remote Repository Files* tells how to configure a GemStone system in which one or more of the repository extents or transaction logs reside on a node remote from the Stone repository monitor.

# 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 via 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, NFS is not required and is 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.

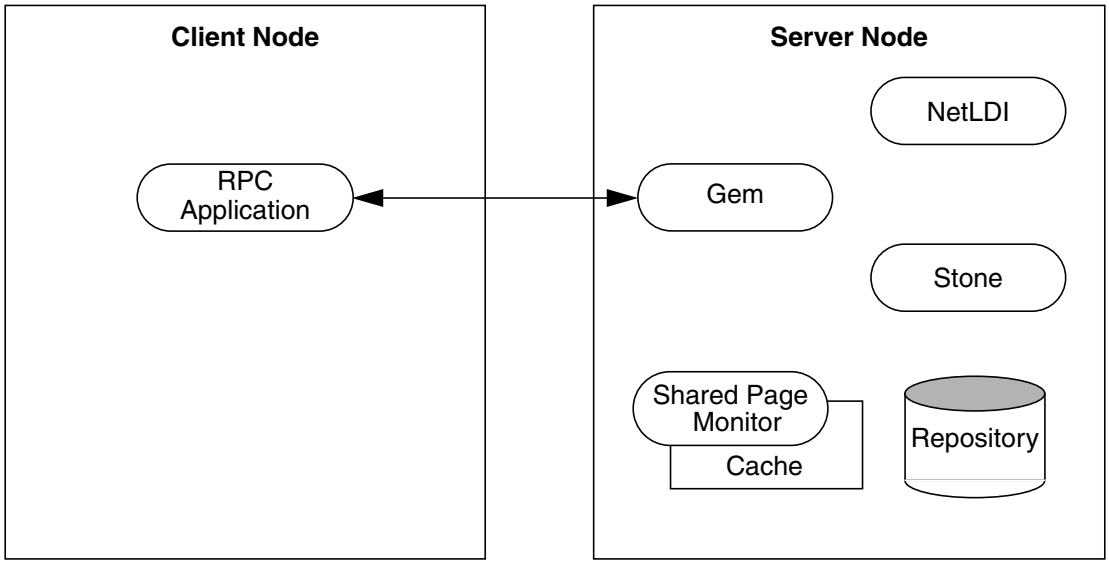
Figure 3.1 shows two typical distributed configurations in which an application on a remote (client) node is logged in to a repository and Stone repository monitor running on a server node.

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 3-23.

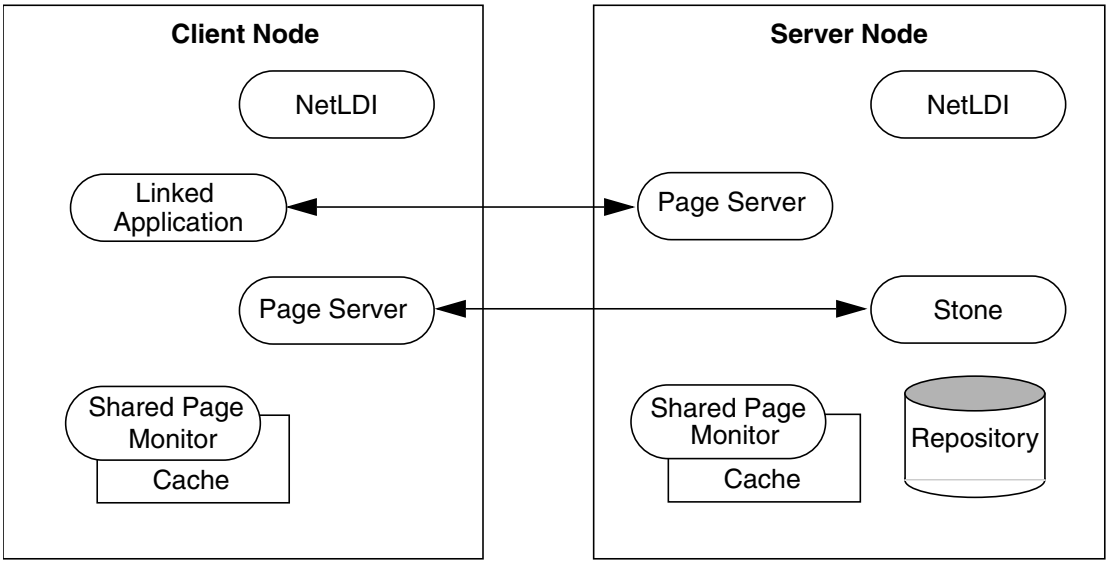
At the bottom of Figure 3.1, the application and the Gem are linked in a single process that runs on the Client node. This configuration avoids the overhead of RPC calls, but in some applications it may increase network traffic substantially if large objects must be brought across the network. Ordinarily, 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 3-19.

**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" that hold 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, or where a repository extent resides, 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 1-31. 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 3-13.

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 `netldi50`. 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 3-15.

## GemStone Page Servers

Remote GemStone repository I/O is carried out by page server processes. The name of the executable file is `pgsvrmain`. For each process that connects to a repository extent across the network (that is, for the repository monitor and each session process), the NetLDI service spawns a `pgsvrmain` on the node where the

extent resides. GemStone never uses NFS (network file system) for repository access.

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

## GemStone Network Objects

GemStone uses the concept of *network object* to encompass the services that a NetLDI can provide to a client. One of these is the page server, which was just discussed. Other network objects include two requested by the Stone at start up: the shared page cache monitor and the Garbage Collector 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/bin/services.dat`. There are two entries for the standard Gem session process:

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

For instance, when you enter "gemnetobject" as a session login parameter (such as for GemNetId in Topaz), the NetLDI maps the request to the script `$GEMSTONE/bin/gemnetobject`. Similarly, an object name can be entered while setting up a GemBuilder session (as Name of Gem Service) or other application. Application programmers provide the name as a parameter to `GciSetNet()` or `GS_System::login()`.

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 | performOnServer:`, which can take environment variables as arguments.

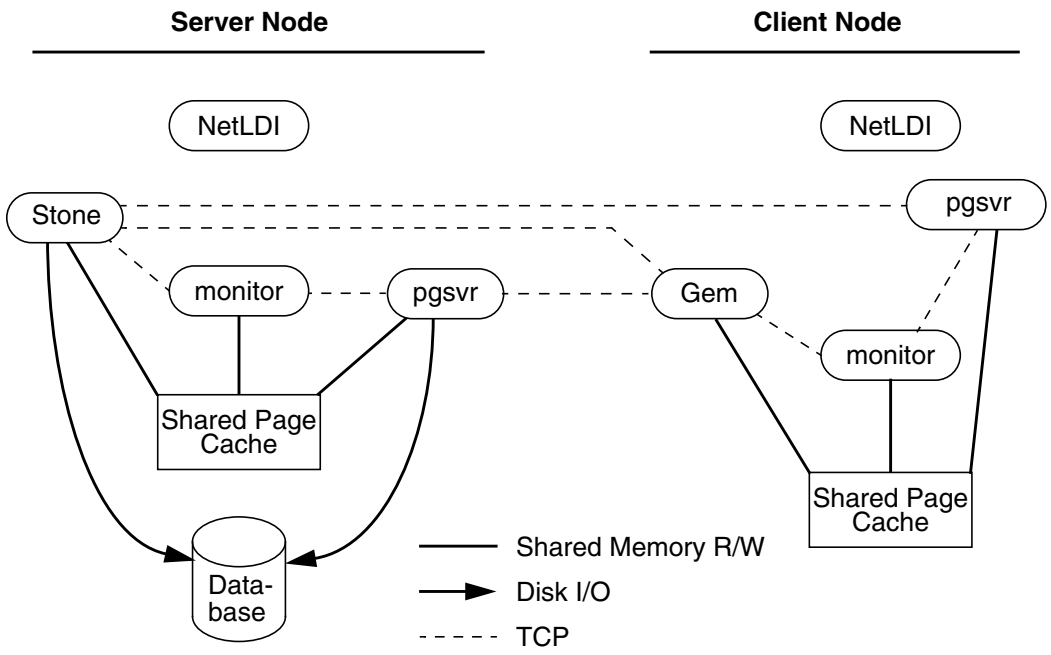
If your application uses a custom Gem executable, you can edit `services.dat` to include the appropriate mapping. For the procedure, see "How to Install a Custom Gem" on page 2-16.

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

A shared page cache can reduce I/O overhead for Gem session processes on a client node, 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**



Whether a shared page cache is created on a remote node depends on several factors in addition to the GEM_SHR_PAGE_CACHE_ENABLED configuration option:

- At start up a shared page cache is created on each node that has a repository extent or replicated extent, and all Gem session processes logged in from that node must use it.
- If a node does not have a repository file for a particular Stone, whether a cache is created depends on the configuration of the first Gem session processes started on that node. If the first Gem creates a shared page cache, other Gems on that node that subsequently log in to the same Stone must use the shared page cache. If the first Gem has the shared page cache disabled, subsequent Gems on that node cannot create one.

If a Gem's configuration file has the cache disabled when these rules require that an existing cache be used, a warning is issued during login.

## 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.

If you want to change the default keepalive interval or to disable it, the *GemStone Installation Guide* explains how to do that for your platform. Be aware that the change affects all network communications on that node.

## 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 3-4). 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 B-16.

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

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

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

## Default: Password Authentication

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

- create a `.netrc` file containing the name of the other node, the login name, and the password,

- 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 and C++ interfaces provide similar capability to application programmers. For further information, refer to calls described in the *GemBuilder for C* and the *GemBuilder for C++* manuals.

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

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

## Alternative: Guest mode with a Captive Account

The NetLDI guest mode can best be combined with captive account mode (page 3-4) 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 | performOnServer: .`

The captive account mode differs from the setuid method described on page 1-31 in that captive account mode affects all services started by the NetLDI, including any ad hoc processes, which are processes started from the user's home directory. (The NetLDI looks in the user's home directory if it cannot find a service listed in `$GEMSTONE/bin/services.dat` .) If you prefer, 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 1-31.

**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
```

## Alternative: Kerberos Authentication

GemStone interfaces (Topaz and the GemBuilder) permit you to log in to a GemStone session by providing a Kerberos ticket as authentication. This section explains that process. If you receive a message like "KrbValidateServer: stubbed," the interface or application you are using does not support Kerberos authentication. It may be possible to relink such programs using Kerberos libraries; consult the installation instructions for your interface.

The instructions in this section are for use with the MIT Project Athena distribution of Kerberos. If that distribution is not available at your site, Appendix H tells administrators how to install Kerberos from the GemStone distribution and how to use the tools provided with GemStone.

Ordinarily, Kerberos authentication is an optional alternative to authentication by user name and password. The administrator can require Kerberos authentication by starting the NetLDI using **startnetldi -k**. When a NetLDI is started with **-k**, passwords will not be accepted for authentication. Users must still include the **#krb** modifier in the NRS, as described below.

Using Kerberos requires the user to take two steps:

**Step 1.** Obtain a Kerberos *ticket-granting ticket* on the appropriate node (examples beginning on page 3-19 indicate which node that is). To obtain the ticket, issue the **kinit** command and enter a user name and password that the administrator registered with Kerberos. For example:

```
% kinit
MIT Project Athena (system)
Kerberos Initialization
Kerberos name: chico
Password: shhhhh
```

The ticket you obtain is valid for a specific time period, typically eight hours. If you log out before the ticket expires, you should first use the **kdestroy** command to ensure security.

**Step 2.** Tell GemStone that a Kerberos ticket is being offered by including the **#krb** authorization modifier in the GEMSTONE_NRS_ALL environment variable

(see "To Set a Default NRS" on page 3-15). GemStone ordinarily expects a system login name and password as authentication. The **#krb** modifier tells GemStone to expect a Kerberos ticket instead.

# 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.

You specify a repository file on a remote node by optionally including **#dbf**. For instance:

```
DBF_EXTENT_NAMES = !@oboe#dbf!/user/data/extent0.dbf;
```

or

```
DBF_EXTENT_NAMES = !@120.0.0.4#dbf!/user/data/extent0.dbf;
```

The list of UNIX-level 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, or you can cause all network access requests to be authenticated using Kerberos.

- 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
```

Use the Kerberos modifier **#krb** to tell the NetLDI to expect a Kerberos ticket as user authentication:

(C shell)
```
% setenv GEMSTONE_NRS_ALL #krb
```

or (Bourne or Korn shell)
```
$ GEMSTONE_NRS_ALL=#krb
$ export GEMSTONE_NRS_ALL
```

You can set more than one modifier in the variable. The following example inserts the Kerberos modifier and also sets the **#log** modifier to a template that names log files according to the name of the process (**%N**) and its process id (**%P**), such as `gem18166.log`:

(C shell)
```
% setenv GEMSTONE_NRS_ALL #krb#log:%N%P.log
```

or (Bourne or Korn shell)

```
$ GEMSTONE_NRS_ALL=#krb#log:%N%P.log
$ export GEMSTONE_NRS_ALL
```

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

## 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 ordinarily must have its kernel configured for shared memory the same as on the server node. Although you can disable the use of shared memory for session processes on a client node, we recommend that you leave it enabled, especially where the client node will have more than one session logged in to the repository.
- 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/GemStone5.0`.

**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/GemStone5.0
Client% source $GEMSTONE/bin/gemsetup.csh
```

or (Bourne or Korn shell)

```
$ GEMSTONE=/Server/users/GemStone5.0
$ 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 is on a PC running Windows or on a Macintosh, 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:

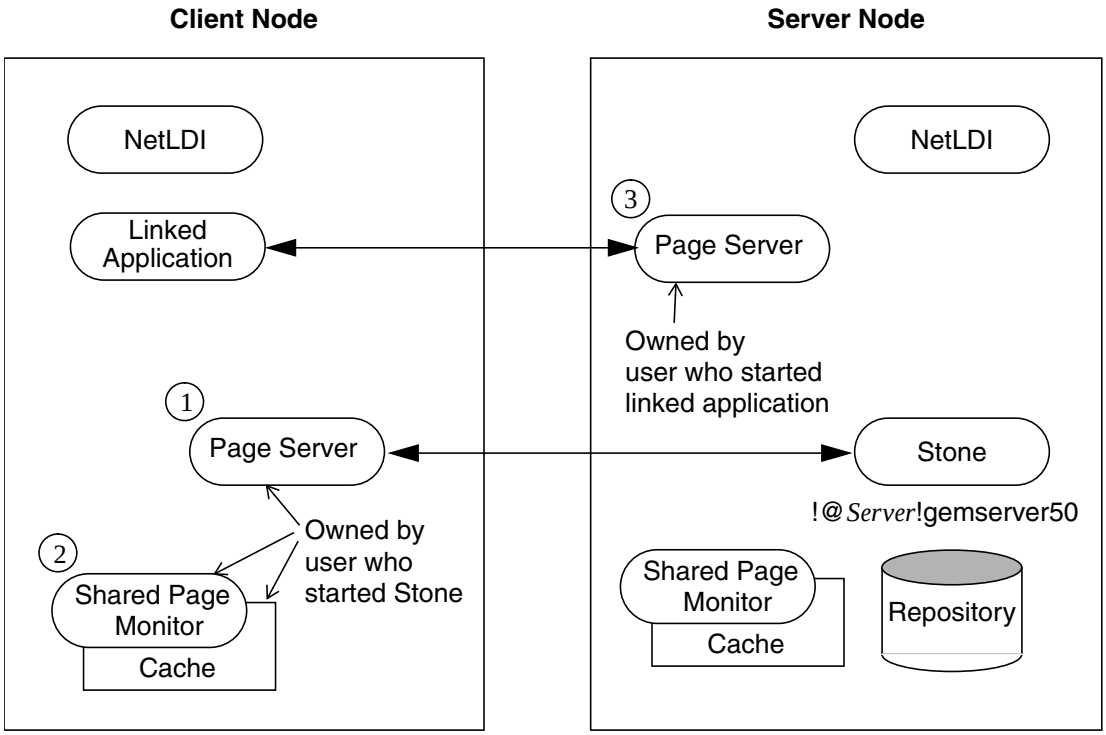
- a linked application connected to a Stone on another node,
- an RPC application with both the session process and the Stone on the server node,
- an RPC application with the session process on the application's node, and
- an RPC application in which all three are on different nodes.

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 3.3 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 3.3 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 2-7.

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/GemStone5.0`.

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

(C shell)

```
Client% setenv GEMSTONE /Server/GemStone5.0
Client% source $GEMSTONE/bin/gemsetup.csh
```

or (Bourne or Korn shell)

```
$ GEMSTONE=/Server/GemStone5.0
$ 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 or Kerberos 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:

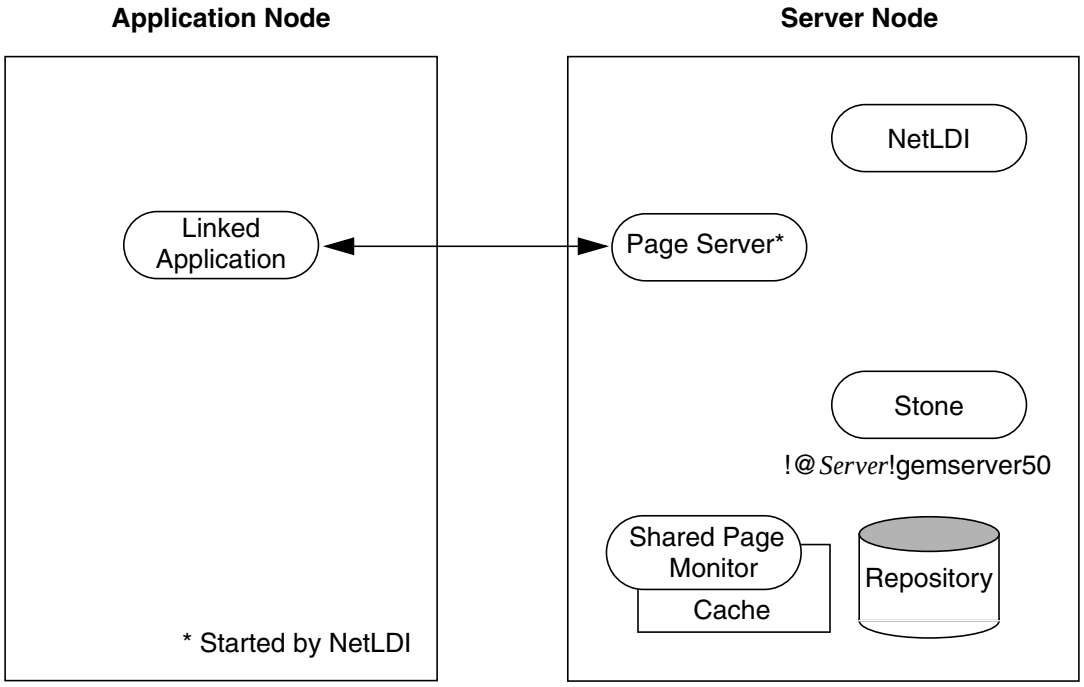
```
Client% topaz -l
topaz> set gemstone !@Server!gemserver50
topaz> set username DataCurator
topaz> login
GemStone Password?
successful login
topaz 1>
```

For comparison, Figure 3.4 shows the same configuration when the shared page cache is disabled on the client node. A NetLDI is not needed on the client because neither the page server nor shared page cache will be started on that node (however, if the cache has been started by another application, this application must use it). The shared page cache is disabled by setting the GEM_SHR_PAGE_CACHE_ENABLED configuration option to false in either a system-wide or an executable-dependent configuration file. For information, see "How GemStone Uses Configuration Files" on page A-2.

*NOTE*

*Although the configuration shown in Figure 3.4 is easier to set up, you should consider it only if a single session process will be logged in to the Stone from that node.*

**Figure 3.4 Linked Application with Gem Shared Page Cache Disabled**




## 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 3-6.

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.

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

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

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

- If your site uses Kerberos, you can obtain a Kerberos ticket on the application node and include the **#krb** modifier in your GEMSTONE_NRS_ALL environment variable (see "To Set a Default NRS" on page 3-15).

**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:

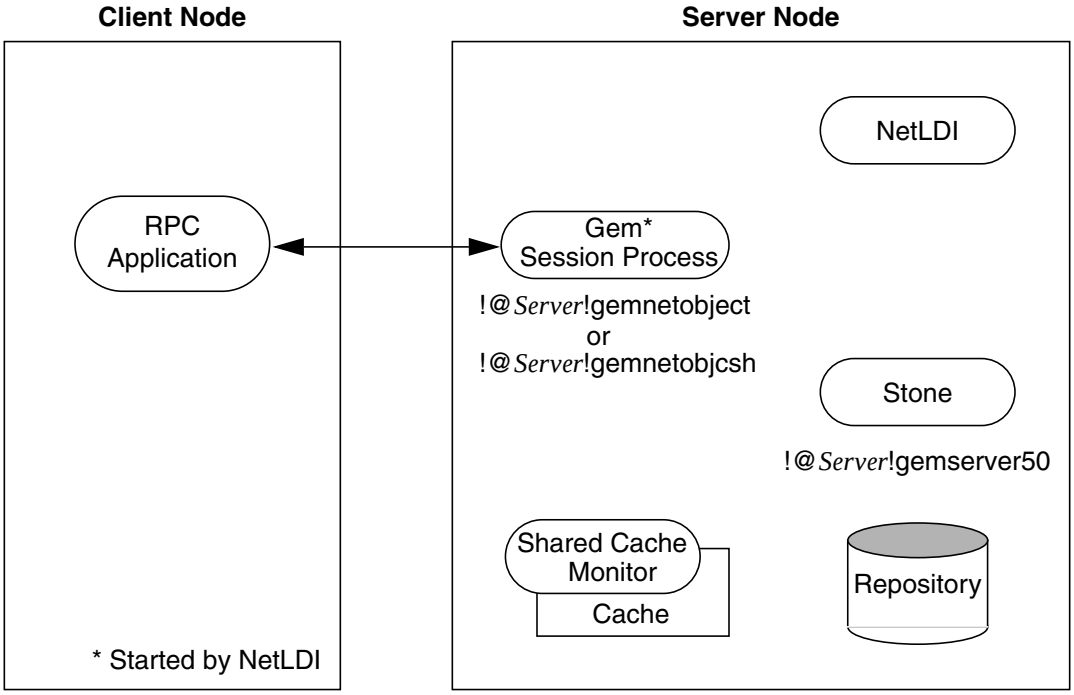
(C shell)

```
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 3-19). 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
  ```

- If your site uses Kerberos, you can obtain a Kerberos ticket on the client node and include the **#krb** modifier in your GEMSTONE_NRS_ALL environment variable (see "To Set a Default NRS" on page 3-15).

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

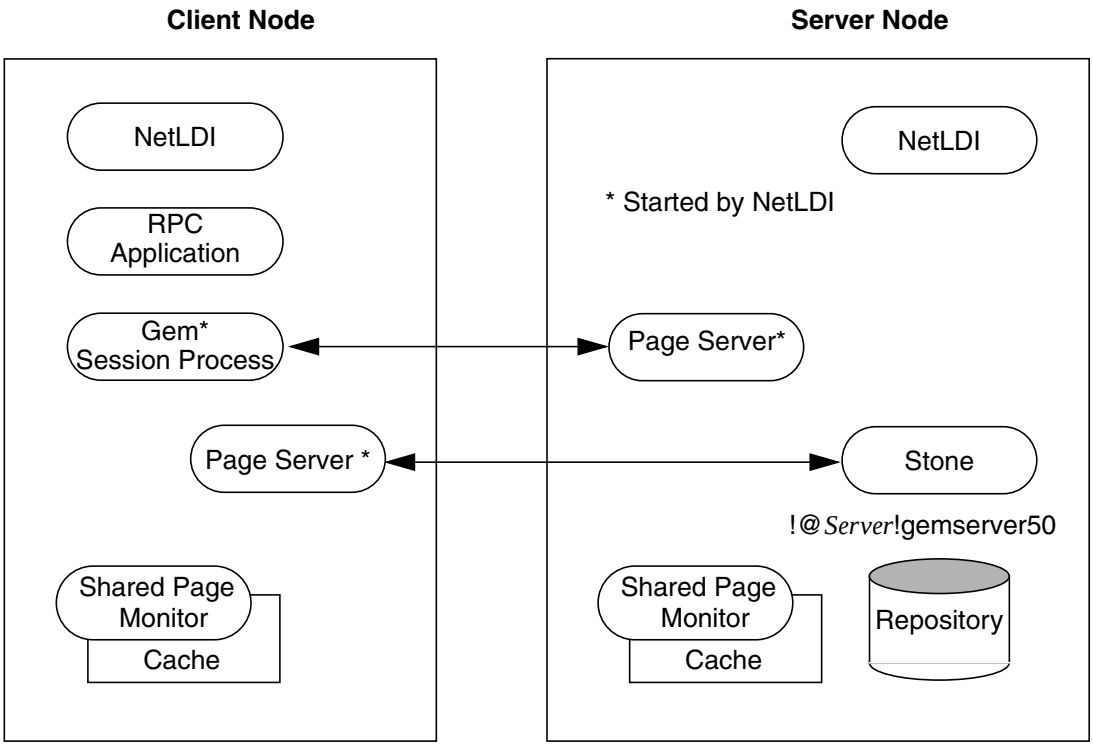
- To start the NetLDI for password or Kerberos authentication, make sure `$GEMSTONE/sys/netldid` is owned by root and has the S bit set. Issue this command (on some operating systems, you may have to issue it as root):

  ```
  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!gemserver50
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.

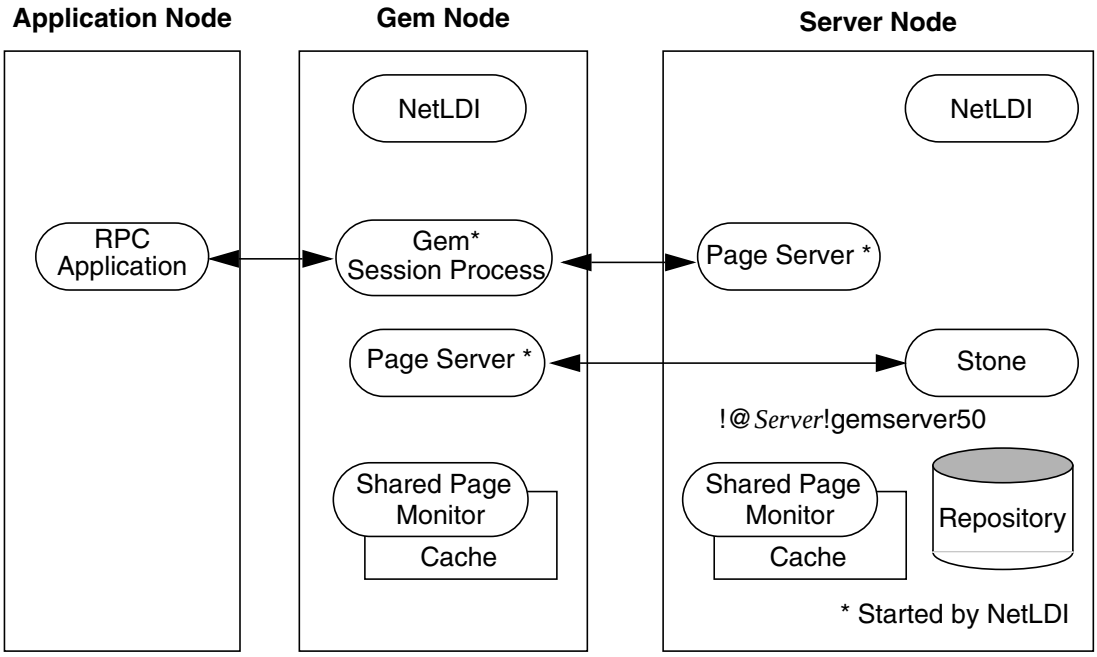
  `machine` *Gem* `login` *userLogin* `password` *secret2*

- 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
  ```

- If you use Kerberos, you can obtain a ticket on the application node. Define your GEMSTONE_NRS_ALL environment variable on the application node to include the **#krb** modifier (see "To Set a Default NRS" on page 3-15).

**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 or Kerberos 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!gemserver50
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 3-11 and 3-13), 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 Oct 18 14:44:45 1994
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 2-7.

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 netldi50. 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/netldi50.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 3-32 for the `topaz` executable.

The user who starts topaz (or the NetLDI captive account when it is in use) must have write permission for $HOME so that the session process can create a log file there. (For a workaround for situations where write permission is not allowed, see “To Set a Default NRS” on page 3-15.)

## Check NetLDI Log Files

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

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

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

# 3.5 How to Set Up Remote Repository Files

This section tells how to set up repository extents and transaction logs, or their replicates, on nodes that are remote from the Stone repository monitor. Although remote replicates can provide protection against single-point media failure, their use may increase susceptibility to network outages because loss of communications or failure of the remote node may bring down the entire object server.

## To Set Up Remote Extents or Replicates

Figure 3.8 shows a configuration in which the Stone runs on Server1, where two extents are located, but a third extent is on another node, Server2. (While this configuration is supported, for best performance it is not recommended.) At start up, the Stone requests the NetLDI on Server2 to spawn a `runpgsvr` process, which in turn starts a page server and shared page cache. For that to happen, the administrator who starts the Stone must have network access to Server2.

*NOTE*

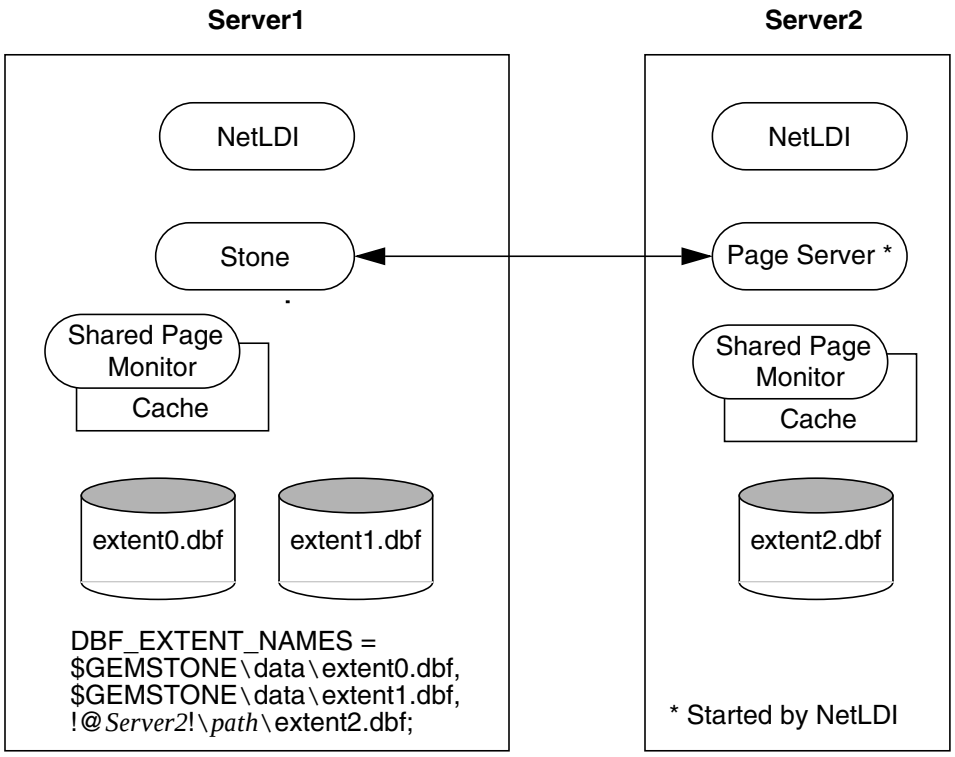
*Each node on which an extent or replicate is located must have its kernel configured to support shared memory.*

Because GemStone is designed for network systems that include heterogeneous platforms, byte order of repository files is not critical. You can use repository files that are not in the native format of their host nodes; the Gem and the Stone automatically determine the format of a repository file and read it correctly. In fact, replicated extents of the same repository may have different byte orders. Replicated extents are created with the native format of the node on which the Stone is running, not the format of the node on which the replicated extent resides.

Byte order does make a slight difference in efficiency. If you have a choice, it is best for the byte order of the repository file to match the native format of the node where the Gems run. However, if Gems are running on heterogeneous platforms,

it is not worth trading off the efficiency of a linked application to keep the Gems in a consistent file format.

**Figure 3.8 Connecting the Stone to a Remote Extent**




If you choose to reverse the byte order of the repository file, see **copydbf** in Appendix B for instructions. That discussion also includes information on the native formats of the various GemStone platforms

In Figure 3.8, any Gem session process running on Server2 directly accesses `extent2.dbf` and the shared page cache on that node. When a Gem on another node needs an object from `extent2.dbf`, the page is read into the shared page cache on Server2 and also into the cache on the Gem's local node. The shared page cache and the Stone's page server on Server2 handle writes to `extent2.dbf`.

*NOTE*

*Be aware that each Gem session process on another node will need to start a page server on Server2 to access the extent and shared page cache.*

*The owner of each Gem process (which may be a captive account) must have an account on Server2.*

Starting a Stone repository monitor with one of its extents located on another node involves several steps:

**Step 1.** Start a NetLDI on both Server1 and Server2 by issuing the **startnetldi** command on each node.

- To start the NetLDI for password or Kerberos 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 -a`*name*

**Step 2.** Decide how you will provide authorization to start the page server on Server2 (for general information, read "How to Arrange Network Security" on page 3-9). There are three choices:

- Your site can run NetLDIs in guest mode, which does not require authentication.

- You can create a `.netrc` file in the home directory on Server1 for the person who starts the Stone repository monitor. That file should contain a line like the following:

  `machine` *server2* `login` *stoneUser* `password` *secret*

- If your site uses Kerberos, you can obtain a Kerberos ticket on Server1 and include the **#krb** modifier in your GEMSTONE_NRS_ALL environment variable:

  (C shell)
  `Server1% setenv GEMSTONE_NRS_ALL \#krb`

  or (Bourne or Korn shell)
  `$ GEMSTONE_NRS_ALL=#krb`
  `$ export GEMSTONE_NRS_ALL`

**Step 3.** Edit the DBF_EXTENT_NAMES configuration option to specify Server2 in a network resource string. For example:

```
DBF_EXTENT_NAMES = $GEMSTONE/data/extent0.dbf,
$GEMSTONE/data/extent1.dbf,  !@Server2!/path/extent2.dbf;
```

**Step 4.** Start the Stone on Server1 by issuing the **startstone** command.

## To Set Up Remote Transaction Logs or Log Replicates

The configuration for remote transaction logs is similar to that shown in Figure 3.8 for remote extents. A NetLDI and page server must be started on the remote node (labeled *Server2* in the figure), and authentication requirements are the same. However, the remote shared page cache and monitor are not needed.

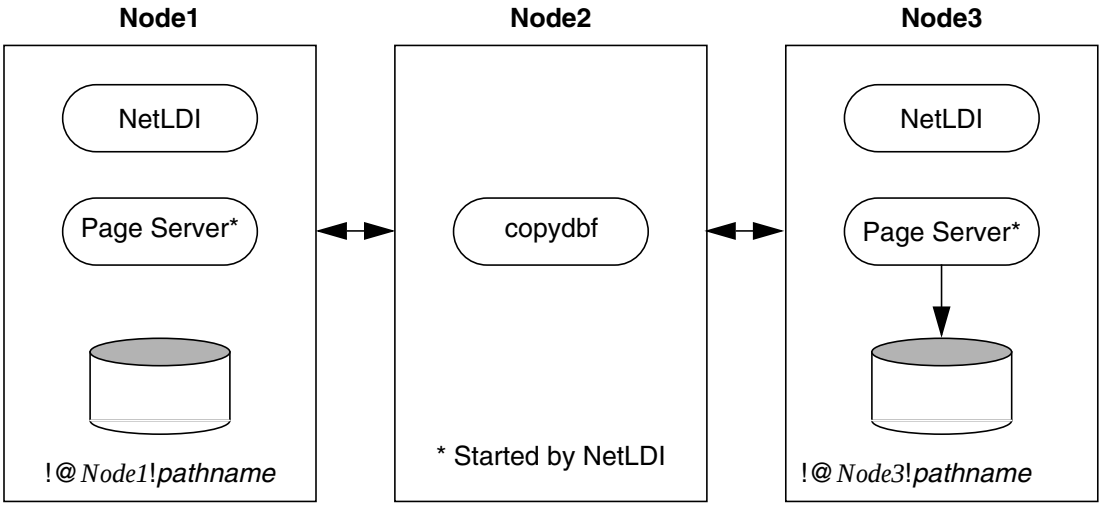
Follow the previous procedure for remote extents with this difference: In Step 3, edit the STN_TRAN_LOG_DIRECTORIES configuration option (or STN_REPL_TRAN_LOG_DIRECTORIES) to specify Server2 in NRS. For example:

```
STN_TRAN_LOG_DIRECTORIES = !@  Server2!/path, !@ Server2!/path;
```

## To Use copydbf Between Nodes

Figure 3.9 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.

**Figure 3.9 Connections for copydbf**



**Step 1.** Unless the NetLDIs are running in guest mode, decide how you will provide authorization for NetLDI services. There are two ways:

- You can create a `.netrc` file in the 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*

- If your site uses Kerberos, you can obtain a Kerberos ticket on Node2 and include the **#krb** modifier in your GEMSTONE_NRS_ALL environment variable (see "To Set a Default NRS" on page 3-15).

**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.

GemStone

# Part II: System Administration

Chapter

# 4 Running GemStone

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

- starting the GemStone Object Server,
- starting Network Long Distance Information (NetLDI) servers,
- logging in to a GemStone session,
- identifying the current sessions, and
- shutting down the object server.

Troubleshooting hints are provided in each startup and login topic. Two additional topics at the end of the chapter explain how to recover from an unexpected shutdown and from a disk-full condition.

## 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/GemStone5.0. The directory $GEMSTONE/bin should be in your search path for commands.

- Which configuration parameters to use — The repository monitor must find a configuration file. The default is $GEMSTONE/data/system.conf. Other files can supplement or replace the default file; for information, see "How GemStone Uses Configuration Files" on page A-2.
- Which repository to use — The configuration file must give the path to one or more repository files (extents) and to space for transaction logs. The default configuration file specifies $GEMSTONE/data/extent0.dbf as the repository file and places the transaction logs in the same directory. You may want to move these files to other locations. For further information, see "Choosing the Extent Location" on page 1-19.

## To Start GemStone

Follow these steps to start GemStone following installation or an orderly shutdown (to recover from an abnormal shutdown, refer to "How to Recover from an Unexpected Shutdown" on page 4-18).

*NOTE*

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

**Step 1.** Set the GEMSTONE environment variable to the *full pathname* (starting with a slash) of the directory where GemStone is installed. Ordinarily this directory has a name like GemStone5.0-sparc.Solaris. For example:

(C shell)

```
% setenv GEMSTONE /users/GemStone5.0-sparc.Solaris
```

or (Bourne or Korn shell)

```
$ GEMSTONE=/users/GemStone5.0-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, and
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 `gemserver50`. For additional information about **startstone**, see the command description in Appendix B.

## 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/gemserver50.log` ), then refer to the discussions that follow this summary.

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

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

## 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 MBytes.
- If the size of the shared page cache has been increased, the operating system's limit on shared memory regions may need to be increased accordingly. Page 1-16 describes a utility (`$GEMSTONE/install/shmem` ) that will help you check the configuration.
- The repository executables (the Stone, Gems, and page servers) must have permission to read and write the shared page cache. Ways to set up access are described in "To Set File Permissions for the Server" on page 1-31. 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/GemStone5.0/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. (If you have extents on a remote node, make sure all NetLDIs are running.)

## 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/GemStone5.0/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/GemStone5.0/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 7-38.
- 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 9-7.
- 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 9-3 and “How to Restore from an Operating System Backup” on page 9-21.

- 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 7-29.

### 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 7-29.) 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 three 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.
- One or more repository extents or transaction log locations reside on a node remote from the Stone.

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 `GemStone5.0-sparc.Solaris`. For example:

(C shell)
```
% setenv GEMSTONE /users/GemStone5.0-sparc.Solaris
```

or (Bourne or Korn shell)
```
$ GEMSTONE=/users/GemStone5.0-sparc.Solaris
$ export GEMSTONE
```

**Step 2.** Use one of the `gemsetup` scripts to set your UNIX path. There is one version for users of the 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 or Kerberos 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 -a`*name*

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

## To Troubleshoot NetLDI Startup Failures

If the NetLDI service fails to start in response to a **startnetldi** command, it's likely that the cause is one of the following. Inspect the NetLDI log (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 NetLDI is to accept only Kerberos authentication but the principal and instance for this NetLDI cannot be found in the Kerberos database (`/etc/srvtab`).
- 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 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, "Launching the Administration Tools." 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 2-2.

## To Define a GemStone Session Environment

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

- Where GemStone is installed—All GemStone users must have a GEMSTONE environment variable that points to the GemStone installation directory, such

as /users/GemStone5.0 -hppa.hpux. 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 "To Set Up the User's Environment" on page 6-10 and "How GemStone Uses Configuration Files" on page A-2.

## To Start a Linked Session

The following steps show how to start a linked application (here, the linked version of Topaz). The steps for setting the GEMSTONE environment variable and the UNIX path for a session are the same as those given on page 4-2 for starting a repository monitor. They are repeated here for convenience.

The procedure assumes that the Stone repository monitor has already been started and has the default name *gemserver50*.

**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 GemStone5.0-sparc.Solaris. For example:

(C shell)
```
% setenv GEMSTONE /users/GemStone5.0-sparc.Solaris
```

or (Bourne shell)
```
$ GEMSTONE=/users/GemStone5.0-sparc.Solaris
$ export GEMSTONE
```

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

GEMSTONE,
GEMSTONE_SYS_CONF,
GEMSTONE_EXE_CONF, 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 *gemserver50* 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 `netldi50`. (This list also shows the Stone and shared page cache monitor.)

```
% gslist
Status Version  Owner   Started  Type Name
------ -------- -------- ------------ ------ ----
exists 5.0    gsadmin  May 24 08:42 cache gemserver50@node3
exists 5.0    gsadmin  May 24 08:42 Stone gemserver50
exists 5.0    gsadmin  May 24 08:42 Netldi netldi50
```

If necessary, start a NetLDI:

- To start the NetLDI for password or Kerberos 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 -a`*name*
  `% 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*

- If your site uses Kerberos, you can obtain a Kerberos ticket and include the **#krb** modifier in your GEMSTONE_NRS_ALL environment variable:

  (C shell)
  `% setenv GEMSTONE_NRS_ALL 5#krb`

or (Bourne or Korn shell)

```
$ GEMSTONE_NRS_ALL=#krb
$ export GEMSTONE_NRS_ALL
```

**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 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 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, the Stone, the GC session and the Stone's AIO page server each use one of this number. On other nodes, the Stone's page server uses one. See "To Set the Page Cache Options and the Number of Sessions" on page 1-13. Check the Stone's log for warnings that the value requested for SHR_PAGE_CACHE_NUM_PROCS has been adjusted to match your system's configuration.
- The UNIX kernel must provide one semaphore for each session that wants to attach to the shared page cache. See "Reviewing Kernel Tunable Parameters" on page 1-12.
- 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 1-12 for the Stone and page 2-5 for Gems. You can examine the kernel limit on some operating systems by invoking **limit** (or in the Bash shell supplied with GemStone, **ulimit -a**).
- 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 servio as of Tue Oct 18 14:44:45 1994
  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 3-30.

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

# 4.4 How to Identify Sessions Logged In

Privileges required: SessionAccess.

To identify the sessions currently logged in to GemStone, send the message `System | currentSessionNames.` This message returns an array of internal session numbers and the corresponding UserId. For example:

```
topaz 1>  printit
System currentSessionNames
%
session number: 1  UserId: GcUser
session number: 2  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*.

The method `System (C)| 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 *GemStone Kernel Reference*.

# 4.5 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: 1     UserId: GcUser
session number: 2     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 `gemserver50`. 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 | 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.*

## 4.6 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, it probably means one of these situations has occurred:

- Normal shutdown
- Disk failure
- Shared page cache 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, and the garbage collection session. The child processes have log names formed from *gemStoneName*, the process id, and a descriptive abbreviation. For instance:

```
gemserver50.log
gemserver5016936pcmon.log
gemserver5016966pgsvraio.log
gemserver5016980gcgem.log
```

Once the problem is identified, your recovery strategy should take into account the interdependence of GemStone processes on different nodes. For instance, if the node with the repository file crashes, the page server dies abnormally. This condition eventually forces an automatic recovery of the repository. To restart, however, you also have to kill the Stone repository monitor, because it has lost its page server. 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:

```
---June 20, 1996 11:25 am Local Daylight Time
SHUTDOWN command received from user DataCurator

Stopping GemStone.
```

After a normal shutdown, restart GemStone in the usual manner. For instructions, see "How to Start the GemStone Server" on page 4-1 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/GemStone5.0/data/extent0.dbf
PageId = 94
File = /users/GemStone5.0/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 7-39.

## 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 monitor log, $GEMSTONE/`data`/*gemStoneName*_`pcmon`*nnnn*`.log`, may indicate the reason.

```
[ShrPc.connfail 1]
     Detected a connection failure from the
SharedPageCache monitor.
     Error Text: 'Protocol = TCP/IP
Reason  = Network Problem
Sys Codes= NONE
TCP error= Network partner disconnected.'
    GemStone Error number = [903:4009]
    "The connection to the shared cache monitor was lost."
```

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 `gemserver50@127.0.0.1` and `gemserver50@127.0.0.1..LCK` for the default Stone name gemserver50.

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 4-1.

## Fatal Error Detected by a Gem

If a Gem session process detects a fatal error that would cause it to halt and dump a core image, the Stone repository monitor may do the same when it is notified of the event. This response on the part of the Stone is configurable through the STN_HALT_ON_FATAL_ERROR option. 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 Stone
    when halt on fatal error was specified in the config file
```

You can change this response by setting the STN_HALT_ON_FATAL_ERROR configuration option to False. That setting causes the Stone to attempt to keep running if a Gem encounters a fatal error; it is the recommended setting for GemStone in a production system.

## Some Other Shutdown Message

In the event of other shutdown messages in the GemStone log:

1. Consider whether the shutdown might have been caused by a disk failure or a corrupt UNIX file system, especially if you see an unexpected message such as `Object not found` . If you suspect one of these conditions, start with a page audit of the repository file (see "How to Audit the Repository" on page 7-29).

   ***If the page audit fails***, read the advice under "Disk Failure or File System Corruption" on page 4-19 of this chapter, and consult your operating system administrator.

   ***If the audit succeeds***, continue to the next step.

2. If you don't suspect disk failure or a corrupt file system, try using **startstone** to restart GemStone. For instructions, see "How to Start the GemStone Server" on page 4-1.

3. If the restart fails, you may have to restore the repository (see "How to Restore a GemStone Repository" on page 9-7).

## No Shutdown Message

If the GemStone log doesn't contain a shutdown message, there has probably been a power failure or an operating system crash. In that event, the Stone repository monitor automatically recovers committed transactions the next time it starts. Use **startstone** to restart GemStone. For instructions, see "How to Start the GemStone Server" on page 4-1." See Appendix B for more information about the **startstone** command; or, for on-line documentation, type **startstone -h** or **man startstone**.

# 4.7 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
- to 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 KBytes (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 "How to Manage the Size of the Repository" on page 7-12 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 7-5.

## 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 8-9.

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 5

# Launching the Administration Tools

This chapter tells how to bring up the GemStone tools from which you can perform administrative tasks on an object server. There are two such tools:

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

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

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

# 5.1 The Administrative Accounts

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

*WARNING*

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

A third administrative account, GcUser, is for the special session that logs in to perform the garbage collection tasks. Because this user is logged in automatically by the Stone repository monitor, the primary reason to log in as GcUser yourself is to tune garbage collection parameters that are stored in GcUser's UserGlobals.

# 5.2 Defining Your GemStone Environment

Before you can launch one of the administrative tools, it's necessary to define the session environment in UNIX. That process is the same as the one described for all users on page 4-10. Define the GEMSTONE environment variable and be sure that $GEMSTONE/`bin` is in your path.

# 5.3 Launching the GemBuilder Administration Tools

Performing GemStone system administration from GemBuilder involves these steps, which are described more fully later:

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

   To start GemBuilder, it must have been properly installed in your Smalltalk image according to the instructions in your GemBuilder release notes. You can

use either your working Smalltalk image or a new stock image, such as `visual.im`. For example:

```
% visualworks visual.im
```

2. In the **GemStone** menu, choose one of the tools in **Admin**.
3. When you finish, remember to commit your transaction so that it becomes a permanent part of the repository.

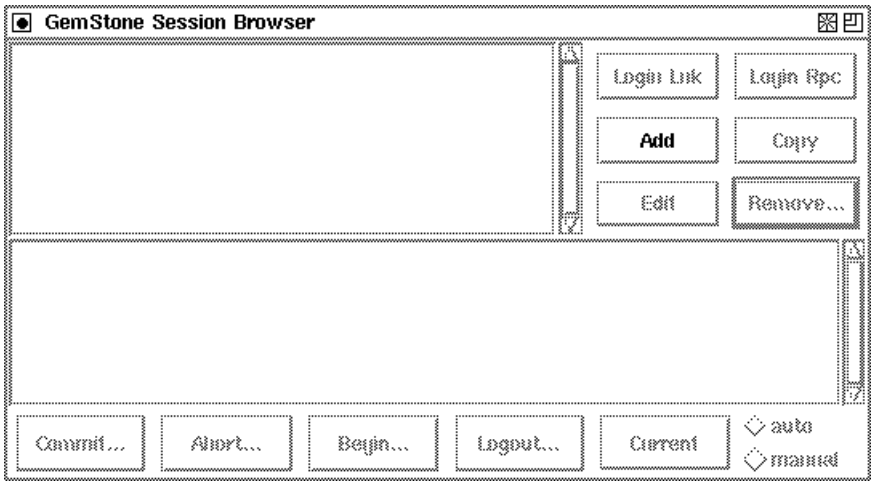
## Logging In Through GemBuilder

Open a GemStone Session Browser (Figure 5.1) by choosing **Tools** > **Session Browser** in the **GemStone** menu.

*NOTE*

*The illustrations shown here are only representative, so the details of yours may be different.*

**Figure 5.1 The GemStone Session Browser**



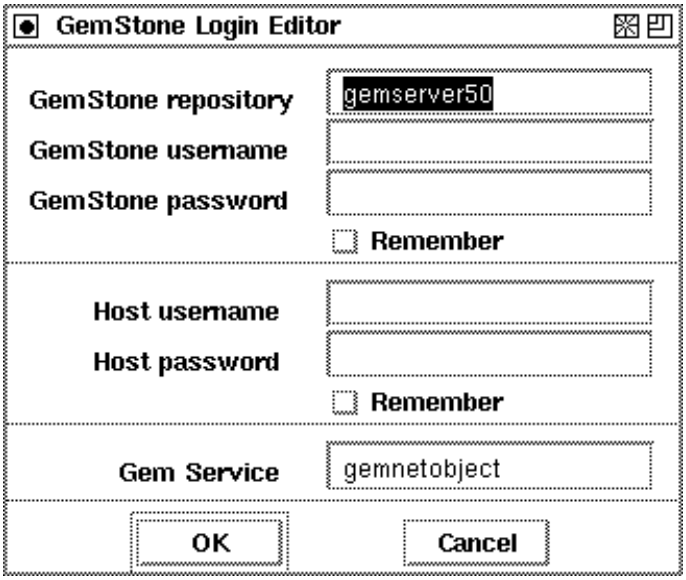
Choose **Add** in the upper right portion of the Session Browser, and then fill in the session parameters in the GemStone Login Editor that appears (Figure 5.2). For example:

| | |
|---|---|
| GemStone repository | `gemserver50` |
| GemStone username | `DataCurator` |
| GemStone password | (*type the DataCurator password*) |

To log in, select the name of the session in the upper left pane of the Session Browser, then choose **Login Lnk**.

For complete information about GemBuilder, see the GemBuilder manual for the vendor Smalltalk you are using.

**Figure 5.2 The Login Editor**



## Finding the GemBuilder Administration Tools

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

Figure 5.3 shows the tools that are available through the **Admin** menu:

- **GemStone Users** lets you examine, modify, and delete existing accounts or create new ones. The GemStone User dialog and the Privileges dialog show the UserProfile for a particular account.
- **Symbol Lists** lets you examine and modify the name space of a particular user. It lets you add and delete dictionaries from the user's SymbolList, as well as examine and modify the entries in the dictionaries that make up the SymbolList.

A third administration tool, the **Segment Tool** in the **Tools** menu**,** lets you create segments and set their authorizations. Segments provide the means for managing GemStone authorization at the object level by assigning objects to segments that have appropriate authorization characteristics.

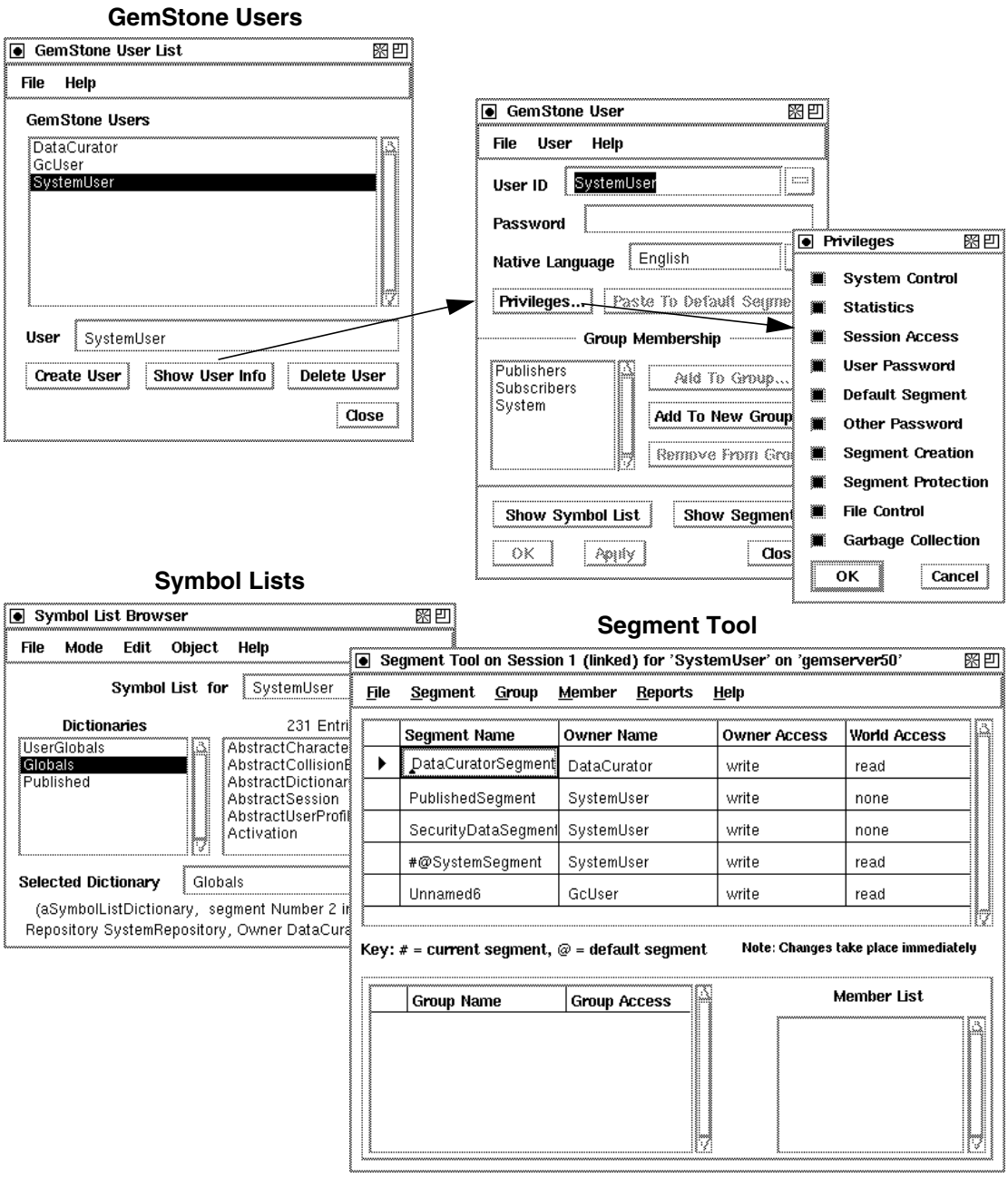
## Committing Your Changes

Remember to commit your changes to the repository. You can do that by choosing **Commit** in the row of buttons at the bottom of the Session Browser. **Commit** is also available in the **File** menu of the administration tools shown in Figure 5.3. If you log out after performing work and do not commit it to the permanent repository, the uncommitted work is lost.

## Logging Out of GemStone

Log out of GemStone from the Session Browser (Figure 5.1) by selecting your session in the browser's lower pane and choosing **Logout** in the row of buttons at the bottom of the browser. When you log out, GemBuilder prompts you to commit your changes.

**Figure 5.3 The GemBuilder Administration Tools**



# 5.4 Invoking Topaz

Performing system administration tasks using Topaz involves three steps:

1. Start Topaz and log in to GemStone.
2. Use the Topaz **printit** command to execute Smalltalk expressions.
3. Use the Topaz **commit** command to make your work a permanent part of the GemStone repository.

*REMEMBER:*
*You must commit your changes to the repository in order to make them permanent.*

## Logging in Through Topaz

To start Topaz, enter:

```
% topaz -l
```

Topaz announces itself with a banner:

```
_____________________________________________________________________________
|       GemStone Object-Oriented Data Management System       |
|              Copyright (C) GemStone Systems, Inc. 1986-1996.                  |
|             All rights reserved.              |
+-------------------------------------------------------------------------+
|  PROGRAM: TOPAZLNK, Linear GemStone Interface (Linked Session)     |
|  VERSION: 5.0, Thu Apr 25 23:02:40 US/Pacific 1996                            |
| BUILT FOR: SPARC (Solaris 2.4)                                                |
| RUNNING ON: 1-CPU mozart sun4m (Solaris 2.5 Generic) 85MHz, 64MB              |
| PROCESS ID: 21577    DATE: Fri 26 Apr 1996 11:02:28 PDT                      |
|_______________________________________________________________________|
topaz>
```

Next, set three parameters that allow you to log in to the repository: the name of your GemStone repository monitor, your GemStone user name, and your

GemStone password. Use the Topaz **set** command to establish these parameters. For example:

```
topaz> set gemstone gemserver50
topaz> set username DataCurator
topaz> set password
GemStone password?  (type your GemStone password)
topaz> login
successful login
topaz 1>
```

The session number in the topaz prompt ("topaz 1>") is a reminder that you are logged in to a GemStone session. If you log in to additional sessions, the number in the prompt shows which session is active.

If you want to exit to your host operating system from Topaz without terminating your GemStone session, use the Topaz **spawn** command. When the operating system prompt appears, you can execute system commands. When you're finished, type **exit**. You can then resume your GemStone work from within Topaz.

## The Printit Command

The **printit** command sends text following the command to GemStone for execution as Smalltalk code, and displays the results. (The **run** command does the same thing.) If there is an error in your code, Topaz displays an error message instead of a legitimate result.

Smalltalk text is terminated by the first line that contains a **%** symbol in column 1. For example:

```
topaz 1>  printit
2 + 2
%
4
```

For complete information about Topaz commands, see the *GemStone Topaz Programming Environment.* You can also use the **help** command at the Topaz prompt.

## The Commit Command

The **commit** command ends the current transaction and stores your changes in the repository.

*NOTE*

*Always remember to commit changes you want to become persistent. Your changes are not part of the repository (and therefore not visible to subsequent sessions) until you issue the Smalltalk message* ***System commitTransaction*** *or invoke the Topaz* ***commit*** *command.*

Chapter

# 6 Administering User Accounts and Security

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

- How to create and modify user accounts, including passwords, privileges, group memberships, and symbol resolution, and how to control the user's read-write access to objects through the use of *segments*.

  There are two versions of these procedures, the first using the GemBuilder administration tools, and the second using Topaz.

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

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

To perform most of these tasks you must either be explicitly authorized to modify an affected segment, or have explicit privilege to execute a restricted Smalltalk method. The overview (next) introduces these concepts. For a full description, see the chapter of the *GemStone Programming Guide* that discusses security.

# 6.1 Overview

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

## UserProfiles

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

The following paragraphs describe each of the elements that you specify when creating a new UserProfile. If you have the necessary privileges or authorization, you can also modify these elements. In addition, the UserProfile contains a symbol list for use in resolving symbols in that user's name space. For a discussion, see "The UserProfile and Session Symbol Lists" on page 6-5.

User ID — Each UserProfile is associated with a userId—a unique String that identifies the user to the GemStone system at login. Embedded spaces are okay.

Password — The user supplies this password (an InvariantString) for identification purposes at login. This password has no connection with a user's operating system password and should be different. GemStone stores the password in encrypted form in a secure manner. Users must have explicit privilege to change their own passwords—or anyone else's. (See the discussion of privileges below.)

GemStone provides a number of ways to restrict the passwords that a user can choose, and it can record login failures and disable the account if failed attempts persist. For information about changing the default settings, see "How to Configure GemStone Login Security" on page 6-34.

Default Segment — When you add a new user to the GemStone system, you must define the user's default segment—the segment that, by default, determines the read and write authorizations for objects created by the user.

In GemStone Smalltalk, a segment groups objects for purposes of authorization (protection); that is, if you can read or write one of a segment's objects, you can read or write all

of them. The owner of a segment can designate named groups whose members are authorized to read or write objects in that segment. For more information about segments, see the chapter in the *GemStone Programming Guide* that discusses security.

When you create a new UserProfile, GemStone ordinarily creates a new segment in the repository to be the user's default segment. By default, the segment's owner (and no one else) can read and write in the new segment. You (or the new owner) can use the segment authorization protocol to enable other users to read and write in that segment. (For details on how to do this, see "To Change the Authorization of a Segment" on page 6-17 for GemBuilder and page 6-31 for Topaz.)

In some cases, you may want to use an existing segment as the default segment for a new UserProfile. For information, see page 6-18 (for GemBuilder) and page 6-22 (for Topaz).

Privileges

When you create a new UserProfile, you determine whether the new user may perform certain "privileged" system functions that are customarily performed by you, as the GemStone data curator. For example, many of the messages to System require explicit privilege. Table 6.1 lists the Smalltalk methods associated with each GemStone privilege.

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

Groups

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

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

**Table 6.1 Smalltalk Methods with GemStone Privileges**

| Type of Privilege | Privileged Methods |
|---|---|
| SystemControl | GsSession \| stop<br>System \| resumeLogins, shutDown, stopOtherSessions, stopSession:, suspendLogins, changeCacheSlotIoLimit:to: |
| Statistics | System \| stoneStatistics |
| SessionAccess | GsSession \| sessionWithSerialNumber:, serialOfSession:, sessionIdOfSerial:<br>System \| concurrencyMode:, currentSessionNames, descriptionOfSession:, stopOtherSessions, userProfileForSession: |
| UserPassword | UserProfile \| oldPassword:newPassword: |
| DefaultSegment | UserProfile \| defaultSegment: |
| OtherPassword | UserProfile \| activeUserIdLimit, activeUserIdLimit:, clearOldPasswords, isDisabled, lastLoginTime, lastPasswordChange, loginsAllowedBeforeExpiration, loginsAllowedBeforeExpiration:, password:, reasonForDisabledAccount,reasonForDisabledAccount:,userId:<br>UserProfileSet \| findDisabledUsers,findProfilesWithAgingPassword |
| SegmentCreation | Segment \| newInRepository: |
| SegmentProtection | Segment \| group:authorization:, groupNo:group:authorization:, ownerAuthorization:, worldAuthorization: |
| FileControl | Repository \| abortRestore, addTransactionLog:replicate:size:, commitRestore, continueFullBackupTo:MBytes:, createExtent:, createExtent: withMaxSize:, createReplicateOf:named:, disposeReplicate:, fullBackupTo:, fullBackupTo:MBytes:, restoreFromArchiveLogs, restoreFromBackup:, restoreFromBackups:, restoreFromCurrentLogs, restoreFromLog:, restoreStatus, setArchiveLogDirectories:...replicatePrefix:, shrinkExtents, startNewLog, timeToRestoreTo: |
| GarbageCollection | Repository \| auditWithLimit:, findDisconnectedObjects, markForCollection, markGcCandidates, objectAudit, pagesWithPercentFree:, reclaimAll, repairWithLimit:, scavengePagesWithPercentFree: |
| (various) | System \| configurationAt:put: |

For a more general discussion of UserProfiles in GemStone, see the discussion of sessions and UserProfiles in the *GemStone Programming Guide.* Also see the instance protocol for UserProfile and UserProfileSet in the *GemStone Kernel Reference*.

## Predefined Users

When GemStone is first installed, the AllUsers object (a UserProfileSet) has UserProfiles already defined for the following users.

SystemUser — The SystemUser account is the owner of the SystemSegment, which contains the kernel classes. This account ordinarily is used only to perform GemStone system upgrades. DO NOT use this account for ordinary administration tasks. Initially, SystemUser has all privileges and belongs to all predefined groups.

DataCurator — The DataCurator account is the account you should use for day-to-day administration tasks. Initially, DataCurator is granted all privileges and belongs to all predefined groups. All GemStone UserProfiles are part of the DataCurator Segment.

GcUser — The garbage collector user is a special account that logs in to the repository to perform garbage collection tasks. Initially, GcUser has only the GarbageCollection privilege and belongs only to group Subscribers.

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

## The UserProfile and Session Symbol Lists

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

Each session's symbol list consists of two parts. A persistent part (an instance of class SymbolList) is stored in the repository and referenced from the UserProfile as the symbolList instance variable. A transient copy of the symbol list is stored in the GsCurrentSession object for a particular logged-in session. A session's transient copy can be modified without affecting (or causing concurrency conflicts with) either the persistent symbol list or the transient copies controlling other sessions. Changes to your own UserProfile's persistent symbol list also change the

symbol resolution of your current session. However, changes to the persistent symbol list are likely to cause concurrency conflicts with other sessions logged in under the same userId. For further information about symbol lists and the GsCurrentSession object, refer to the *GemStone Programming Guide*.

## The UserGlobals SymbolDictionary

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

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

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

## The Globals SymbolDictionary

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

## The Published SymbolDictionary

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

## Sharing Objects

As described above, the Globals dictionary provides all GemStone users with access to such objects as the kernel classes Integer and Collection. If you want GemStone users to share other objects as well, you need to arrange for references to those objects to be added to the users' symbol lists. There are three primary ways to do this:

- As a member of group Publishers, you can add the objects to the Published dictionary. This dictionary is already in each user's symbol list, so whatever you add becomes visible to users the next time they obtain a fresh transaction view of the repository. If you are using the GemBuilder administration tools, the procedure is similar to that described on page 6-13 for copying a dictionary, except that you select an object in the **Entries** pane and choose **Edit > Copy Entry**. If you are using Topaz, send the message `Publishedat:` *aKey* `put:` *aValue*.
- You can have users add a special dictionary, such as an application dictionary, to their own symbol list. The procedure is described under "To Add a Dictionary to a Symbol List" on pages 6-13 and 6-26. DataCurator typically can't make this change for another user without first obtaining write authorization in that user's DefaultSegment.
- 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 Programming Guide*.

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

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

# 6.2 How to Use the GemBuilder Administration Tools

This section tells how to use the GemBuilder administration tools. There are two subsections: Administering User Accounts and Administering Segment Authorization.

There is a general introduction to the GemBuilder tools in Chapter 5, "Launching the Administration Tools." For complete information, see the *GemBuilder* manual for the version of Smalltalk you are using.

## Administering User Accounts

The GemBuilder administration tools help you create user accounts (UserProfiles) and examine, modify, and remove existing accounts. Most of these tasks use the GemStone User List (left side of Figure 6.1) to open a GemStone User dialog (right side) for a particular user. Follow these general steps for most tasks described in this section:

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

**Step 2.** When the user list appears, select the UserId and choose **Show User Info**.

**Step 3.** When the GemStone User dialog appears, carry out the steps in one or more of the specific procedures that follow.

**Step 4.** Click on **OK** or **Apply** when you are finished.

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

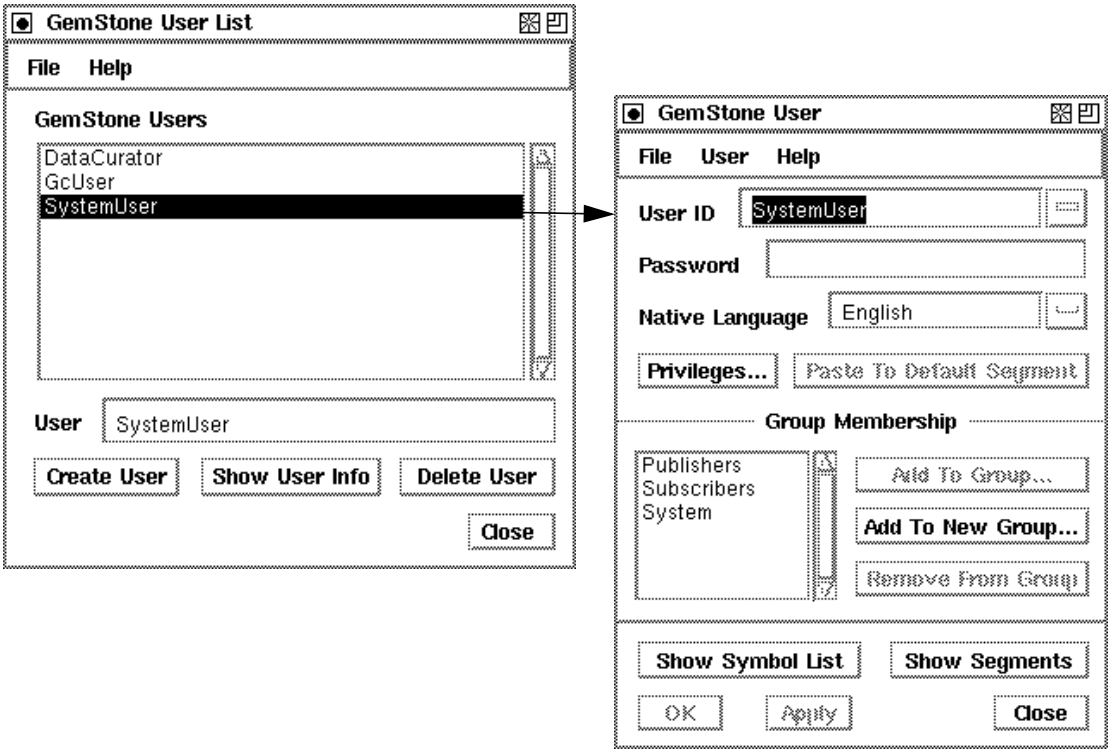
### To List Existing Users

Privileges required: None.

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

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

**Figure 6.1 GemBuilder Tools for Accessing User Profiles**




## To Add a User

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

When you add a user, the GemStone User dialog creates a new UserProfile object and stores it with other UserProfiles in the global object AllUsers. A new segment with world read permission is created for the user and is assigned as that user's current segment. By default, the new UserProfile has no privileges and no group membership.

**Step 1.** In the **GemStone** menu, choose **Admin > GemStone Users**. When the list appears, choose the **Create User** button.

**Step 2.** When an empty GemStone User dialog appears, enter the **User ID**.

**Step 3.** If you enter a **Password**, it will not be echoed, but you will be asked to verify it. If you do not enter a password, it will be set to "gemstone". The

password may not be the same as the UserId and may not be longer than 1024 characters.

**Step 4.** You can also set privileges and group membership at this time. These operations are described on page 6-12 and page 6-14, respectively.

**Step 5.** Choose **OK** when you are finished, or choose **Apply** if you want to add another user.

**Step 6.** Choose **Commit** in the **File** menu.

**Step 7.** Set up the user's environment, as explained next.

## To Set Up the User's Environment

In addition to adding the UserProfile, you should make sure that each new user has the GEMSTONE environment variable defined so that it points to the installation directory. Each user also must know how to execute `$GEMSTONE/bin/gemsetup` as described under "To Start a Linked Session" on page 4-11 or perform similar functions within an initialization file.

Users may need to belong to the same UNIX group as the Stone repository monitor process so that they can access the GemStone extents and shared page cache. For further information, see "To Set File Permissions for the Server" on page 1-31.

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

- GEMSTONE_SYS_CONF and GEMSTONE_EXE_CONF, which can point to customized configuration files for specific GemStone servers and user sessions. For general information, see "Creating or Using a System Configuration File" on page A-5.
- GEMSTONE_NRS_ALL, which can set a number of network-related defaults, including the type of user authentication that GemStone expects. For further information, see "To Set a Default NRS" on page 3-15.
- GEMSTONE_LOG, which can set the location of system log files for the Stone repository monitor and its child processes. For further information, see "GemStone Server Logs" on page 10-2.
- GEMSTONE_LANG, which can be 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 G-1.

- GEMBUILDER, which points to the directory where GemBuilder is installed. See the documentation for your version of GemBuilder for details about the need for this or other environment variables.

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

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
```

## To Remove a User

Privileges required: write authorization in the DataCurator Segment.

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

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

**Step 2.** When the user list appears, select the UserId.

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

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

## To Change a Password

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

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

Your choice of passwords for your own account may be subject to optional restrictions as to pattern and the use of certain words. For information, ask your data curator or see "How to Configure GemStone Login Security" on page 6-34.

**Step 1.** In the **GemStone** menu, choose **Admin > GemStone Users**. When the user list appears, select the UserId and choose **Show User Info**.

**Step 2.** Enter the new password in the GemStone User dialog, then press Tab or Return. The password may not be the same as the UserId and may not be longer than 1024 characters.

**Step 3.** When the verification window appears, enter the new password again.

**Step 4.** Choose **OK** or **Apply**. If you are changing your own password, you will be asked to enter the old password.

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

## To Change a User's Privileges

Privileges required: write authorization in the DataCurator Segment.

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

**Step 1.** In the **GemStone** menu, choose **Admin > GemStone Users**. When the user list appears, select the UserId and choose **Show User Info**.

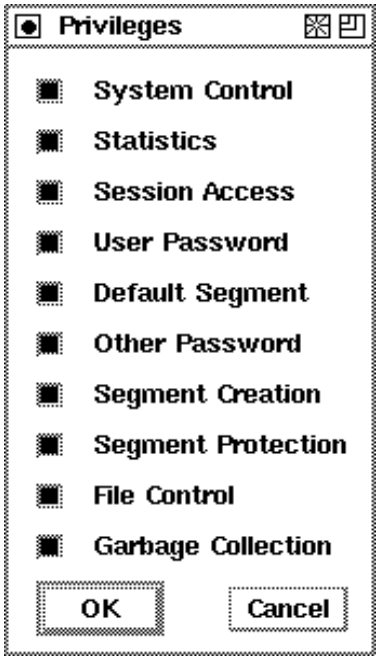
**Step 2.** Choose the **Privileges** button in the GemStone User dialog. Figure 6.2 shows the resulting Privileges dialog.

**Step 3.** When the Privileges dialog appears, click on the check boxes to toggle each privilege that you want to change.

**Step 4.** Choose **OK** when you are finished.

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

**Figure 6.2 The Privileges Dialog**




## To Add a Dictionary to a Symbol List

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

You can copy a dictionary from one symbol list and add it to a different user's persistent symbol list by using the Symbol List Browser. The change does not affect the transient copy of the symbol list that is used by another currently logged in session until that session commits or aborts.

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

**Step 2.** When the Symbol List Browser appears, open **Symbol List for** and choose a userId that already has the dictionary in its symbol list. See Figure 6.3.

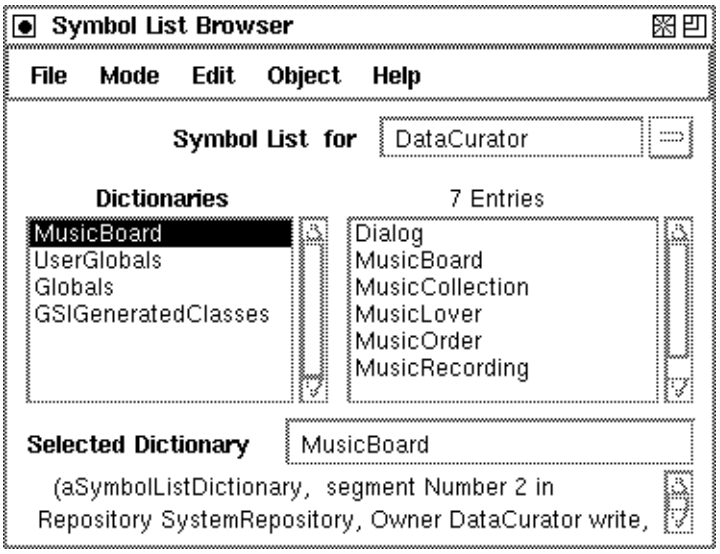
**Step 3.** In the **Dictionaries** pane, choose the dictionary you want to copy. Then choose **Edit > Copy Dict**.

**Step 4.** Reopen **Symbol List for** and choose the userId where you want to add the dictionary.

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

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

**Figure 6.3 The Symbol List Browser**




## To Examine a User's Group Memberships

No privileges are required for this operation.

- The memberships for a particular user are listed in the GemStone User dialog. In the **GemStone** menu, choose **Admin > GemStone Users**. When the user list appears, select the UserId and choose **Show User Info**.
- For information about group memberships for all users, first open the Segment Tool by choosing **Tools > Segment Tool** in the **GemStone** menu. Then choose **Reports > Group Report** to see a list of users in each group, or choose **Reports > User Report** to see a list of groups for each user.

## To Add a User to a Group

Privileges required: write authorization in the DataCurator Segment.

You can use the GemStone User dialog to add group membership for a particular user:

- If the group already exists in the repository, choose **Add To Group** in the GemStone User dialog. Then choose the group name in the list that appears.
- If the group is a new one in the repository, choose **Add To New Group** in the GemStone User dialog. Then enter the name in the dialog box that appears.

To add group membership for several users, it's easier to use the Segment Tool:

**Step 1.** Open the Segment Tool by choosing **Tools > Segment Tool** in the **GemStone** menu.

**Step 2.** Select the target group:

- ❒ If the group already exists in the repository, first select a segment for which the group has access and then select the group's name in the list of groups that appears.
- ❒ If the group is a new one, first select the segment to which it is being given access. Then create the group by choosing **Group > Add**. Enter a name in the box that appears and choose **OK**. By default, the new group will have read access to that segment; to give write access, select the current access and change it to "write".

**Step 3.** Choose **Member > Add** and then enter an existing UserId in the box that appears. Choose **OK**. Repeat this step for each user.

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

## To Remove a User from a Group

Privileges required: write authorization in the DataCurator Segment.

You can use the GemStone User dialog to remove group membership for a particular user. Select the group name in the list of group memberships. Then choose **Remove From Group** and either **OK** or **Apply**.

To remove several users from the same group, it's easier to use the Segment Tool:

**Step 1.** Open the Segment Tool by choosing **Tools > Segment Tool** in the **GemStone** menu.

**Step 2.** Select the group by selecting a segment for which the group has access and then selecting the group name.

**Step 3.** Select a UserId you want to remove, and then choose **Member > Remove**. Repeat this step for each user.

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

### To List All Members of a Group

No privileges are required for this operation.

**Step 1.** In the **GemStone** menu, choose **Tools > Segment Tool**. (You can also reach this tool from a GemStone User dialog by choosing the **Show Segments** button.)

**Step 2.** When the Segment Tool appears, choose **Group Report** from the **Reports** menu.

## Administering Segment Authorization

This section tells how to administer segments using GemBuilder's Segment Tool.

### To Find Out Who Is Authorized to Read or Write in a Segment

Privileges required: read authorization for the segment with which this segment itself is associated, such as the DataCurator Segment.

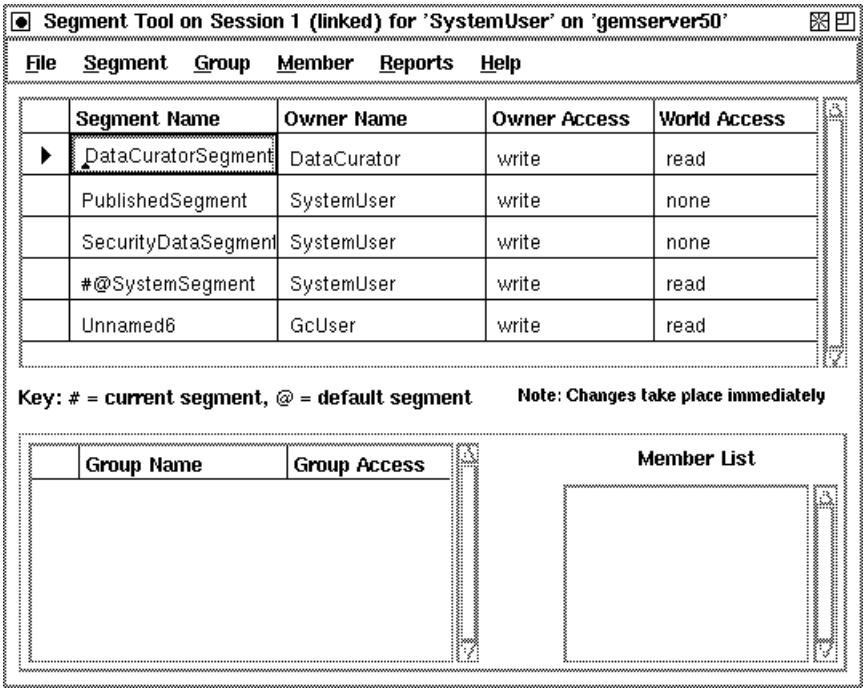
Each segment maintains access authorization for its owner, the world, and an unlimited number of groups. There are three levels of access: none, read-only, and write (which includes read access).

**Step 1.** In the **GemStone** menu, choose **Tools > Segment Tool** to bring up the Segment Tool, Figure 6.4. (You can also access this tool by choosing **Show Segments** in a GemStone User dialog.)

**Step 2.** In the Segment Tool, choose **Reports > Segment Report**. The report lists the owner, world, and group authorizations for each segment.

**Step 3.** To view the members of a particular group, choose **Reports > Group Report**. To view the groups to which each user belongs, choose **Reports > User Report**.

**Figure 6.4 The Segment Tool**




## To 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 levels of access: none, read-only, and write (which includes read access).

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

**Step 1.** In the GemStone menu, choose **Segments** to bring up the Segment Tool, Figure 6.4. (You can also access this tool by choosing **Show Segments** in a GemStone User dialog.)

The top half of the dialog shows the owner, the owner's access, and the world access for each segment in the repository.

**Step 2.** To change owner or world access, select the existing permission you want to change. Then enter a new permission ("read", "write", or "none").

*CAUTION*

*Use caution when removing write authorization while a user is logged in. The user will be unable to commit changes if write authorization is removed from the current segment, and if it is the user's default segment, the user's session will be terminated and the user will be unable to log in again.*

**Step 3.** To set up or change group access to this segment, do the following:

- ❒ Make sure the segment is selected in the top half of the tool. If necessary, click in the first column of the segment's row.
- ❒ To add a group to the authorization list for this segment, choose **Add** from the **Group** menu. Enter the group name in the dialog box that appears. If the group does not exist in the repository, you will be asked to confirm its creation.
- ❒ To remove a group from the authorization list, first select the group by clicking in the first column of the **Group Name** list. Then choose **Remove** from the **Group** menu. You will asked to confirm the action.
- ❒ To change the type of access for a particular group, first select that group in the **Group Name** list and select the existing permission. Then enter the new permission ("read" or "write").
- ❒ To add a member to a group that has access to this segment, first select that group in the groups list. Then choose **Add** from the **Member** menu. Enter the UserId and choose **OK**. (A UserProfile with that UserId must already exist in the repository.)
- ❒ To remove a member from a group that has access to this segment, select the UserId in the member list and choose **Remove** from the **Member** menu. You will be asked to confirm the action.

**Step 4.** Remember to commit your transaction before logging out. A convenient way to do that is by choosing **Commit** from this tool's **File** menu.

## To Change a User's Default Segment

Privileges required: DefaultSegment to change your own; 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 your own default segment:

**Step 1.** Open the Segment Tool by choosing **Tools > Segment Tool** in the **GemStone** menu.

**Step 2.** Select the desired segment by clicking in its first column. Then choose **Segment > Make Default**. (Alternatively, you can type the @ symbol in front of the segment name.)

To change someone else's default segment:

**Step 1.** In the GemStone User dialog, choose **Show Segments**.

**Step 2.** When the Segment Tool appears, select the desired segment by clicking in the first column. Choose **Segment > Grab**.

**Step 3.** Return to the GemStone User dialog. Choose **Paste To Default Segment**.

**Step 4.** Choose **OK** or **Apply**.

*NOTE*

*If you change any user's default segment (including your own) to a segment for which that user lacks write authorization, and you subsequently commit the transaction, the affected user will no longer be able to log in to GemStone.*

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

# 6.3 How to Use Topaz as an Administration Tool

This section tells how to use the Topaz Programming Environment to administer user accounts and segment authorizations. There is a general introduction to Topaz in Chapter 5, "Launching the Administration Tools."

For further information about the Smalltalk expressions in this section, refer to the *GemStone Kernel Reference*.

## Administering User Accounts

This section explains how to create user accounts (UserProfiles) and how to examine, modify, and remove existing accounts using Topaz. Most of these tasks explicitly access UserProfiles in the global object AllUsers.

### To List Existing Users

Privileges required: None.

There is no direct method within GemStone Smalltalk to list only the names of existing accounts. The following example shows one way to obtain that information:

```
topaz 1>  level 1
topaz 1>  printit
AllUsers collect: [ :each | each userId ] .
%
an IdentityBag
  _varyingSize  3
  _numEntries  3
  _indexedPaths  nil
  _levels  0
  #1 DataCurator
  #2 SystemUser
  #3 GcUser
```

## To 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. **You must add each new user to AllUsers**.

In addition to creating the new UserProfile, you should also see that each user's UNIX environment is set up to provide access to GemStone. That discussion is on page 6-10.

GemStone Smalltalk provides two methods for adding a user. The simplified form requires only a UserId and a password. The complete form also allows you to set the user's default segment, and the privileges and groups for that segment.

### To Use the Simplified Form (No Privileges or Groups)

You can use this "streamlined" approach 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
%
```

### To Use the Complete Form (Assign Privileges and Group Memberships)

This approach 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 discussion entitled "UserProfiles" on page 6-2.

## To 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 "How to Configure GemStone Login Security" on page 6-34.

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
%
```

## To Change Another User's Password

Privileges required: OtherPassword.

To modify the password of a GemStone user (other than your own), execute the following expression.

```
topaz 1> printit
(AllUsers userWithId: 'theUserId')
      password: 'newPasswordString' .
System commitTransaction
%
```

The new password takes effect when you commit the current transaction.

The password set by this method is not subject to the optional pattern restrictions described on page 6-34 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.

## To Examine a User's Privileges

No privileges are required for this operation.

GemStone provides messages that allow you to determine which privileged methods a GemStone user may execute, and to change the privileges of any user. Naturally, you need the appropriate 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 6.1 on page 6-4 for a list of the Smalltalk methods that correspond to each privilege.

## To 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' .
```

## To 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' .
```

## To 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
%
```

## To 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.

## To Examine a User's Group Memberships

No privileges are required for this operation.

To find out which groups a user belongs to, first make sure that the Topaz display level is sufficient to display that information. Use the Topaz **level** command to raise the level to at least 1, so that group membership information will be displayed:

```
topaz 1> level 1
```

Now execute the following expression:

```
topaz 1> printit
(AllUsers userWithId: 'theUserId') groups
%
```

This expression returns a Set of Strings indicating the groups to which the user belongs.

## To Add a User to a Group

Privileges required: 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.

## To 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 Programming Guide*.

## To List All Members of a Group

No privileges are required for this operation.

To list all members of a user group, first make sure that the Topaz display level is sufficient to display that information. Use the Topaz **level** command to raise the level to at least 1, so that group membership information will be displayed:

```
topaz 1> level 1
```

Now execute the following expression:

```
topaz 1> printit
AllUsers membersOfGroup: aString
%
```

This expression returns an IdentitySet containing the userId for each member of the group.

## To Remove a User Group

Privileges required: 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) size == 0
      ifFalse: [ AllUsers removeGroup: theGroup ].
"It's okay to remove the group itself"
AllGroups remove: theGroup .
%
```

## To Modify Someone's User ID

Privileges required: 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.

## To 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.

## Administering Segment Authorization

This section tells how to administer segments using Topaz.

### To Find Out Who Is Authorized to Read or Write in a Segment

Privileges required: read authorization for the segment with which this segment is associated, such as the DataCurator Segment.

Each segment maintains access authorization for its owner, the world, and an unlimited number of groups. There are three levels of authorization: none, read (read-only), and write (which includes read permission).

You can find out who is authorized to read or write objects in a segment by sending it the message `asString`. For instance:

```
topaz 1> printit
PublishedSegment asString
%
aSegment, Number 5 in Repository SystemRepository
Owner SystemUser write
Group Subscribers read
Group Publishers write
World none
```

## To Change the Authorization of a Segment

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

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.

*WARNING:*
*Do not, under any circumstances, attempt to change the authorization of the SystemSegment.*

To change the authorization for a segment, execute any (or all) of the following expressions.

```
topaz 1> printit
theSegment ownerAuthorization: #anAuthorizationSymbol .
%
topaz 1> printit
theSegment worldAuthorization: #anAuthorizationSymbol .
%
topaz 1> printit
theSegment group: #aGroupString
      authorization: #anAuthorizationSymbol .
%
```

*NOTE*
*Exercise caution when changing the authorization for any segment that a user may be using as his or her default segment or current segment — whether or not the user owns the affected segment. If a user attempts to commit a transaction, but has created objects in a segment for which he or she no longer has write authorization, an error will be generated.*

For example, to authorize the group Accounting to read (but not write) in user Eli's default segment, you could execute the following expression:

```
topaz 1> printit
(AllUsers userWithId: 'Eli') defaultSegment
 group: #Accounting authorization: #read .
%
```

If the group #Accounting does not exist, GemStone will return an error. The discussion "Add a User to a Group" earlier in this chapter tells how to create a new GemStone group.

## To Remove a Group from a Segment's Authorization List

Privileges required: SegmentProtection 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
%
```

## To 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.*

## To 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 7-29.")

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.

# 6.4 How to Configure GemStone Login Security

GemStone provides several login security features. You can:

- constrain the choice of passwords to a certain pattern, ban particular passwords altogether, or ban reuse of a password by the same account,
- require users to change their passwords periodically (password aging),
- limit the number of logins under a temporary password,
- disable accounts that have not logged in for a specified interval (account aging),
- limit the number of concurrent sessions by a particular account, and
- 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.

## To 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 6.2 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 6.2 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 6.2). 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
%
```

## To 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 6-12.

## 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, which includes the DateTime at which the password will expire.

## Finding Accounts With Password About to Expire

Privileges required: OtherPassword.

You can find out which accounts have a password within the warning period set by `passwordAgeWarning:`. To do this, send the message `findProfilesWithAgingPassword` to AllUsers. For example:

```
topaz 1> printit
AllUsers findProfilesWithAgingPassword
  collect: [ :u | u userId] .
%
an OrderedCollection
#1 qa1
#2 qa2
#3 qa3
```

## Finding Out When a Password Was Changed

Privileges required: OtherPassword.

You can find out the last time the password was changed for a particular userId by sending the message `lastPasswordChange` to that account's UserProfile. This

example converts the DateTime returned to a particular pattern based on MM/DD/YY:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') lastPasswordChange US12HrFormat
%
03/19/96 11:28 am
```

## To 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 reactive the account by giving it a new password as explained on pages 6-12 and 6-23.

### Finding Out When an Account Last Logged In

Privileges required: OtherPassword.

If at least one age limit applies to an account, you find out when that account last logged in by sending the message `lastLoginTime` to that account's UserProfile. For example:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') lastLoginTime US12HrFormat
%
03/19/96 01:40 pm
```

The time of the last login is maintained only if loginsAllowedBeforeExpiration is set in that UserProfile or if at least one of these instance variables is set in AllUsers: passwordAgeLimit, passwordAgeWarning, or staleAccountAgeLimit.

## To Limit Logins Until Password Is Changed

Privileges required: OtherPassword.

When you assign a password to an account, you can make the password temporary by limiting the number of times it can be used. This limitation applies only to a specific account, that is, to the UserProfile that is the receiver of the message. It is intended for use with a new or reactivated account as a means of ensuring that the user changes the password. For example, the following limits the account "qa2" to two more logins under the current password:

```
topaz 1>  printit
(AllUsers userWithId: 'qa2')
  loginsAllowedBeforeExpiration: 2 .
System commitTransaction
%
```

A setting of 0 (the default) disables this feature.

The limit remains in effect until the user changes the password (see pages 6-12 and 6-23). 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 the same pages.

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

## To Limit Concurrent Sessions by a Particular UserId

Privileges required: OtherPassword.

You can limit the number of concurrent sessions logged in under a particular userId by sending the message `activeUserIdLimit:` *aPositiveInteger* to the

UserProfile for that account. For example, the following limits the userId “qa2” to four concurrent sessions:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') activeUserIdLimit: 4 .
System commitTransaction
%
```

A setting of 0 (the default) disables this feature.

If a user attempts to log in when the maximum number of sessions for that userId are already logged in, the login is denied and the fatal error #gsActiveUserLimitExceeded is returned.

# To Record Login Failures

The Stone repository monitor keeps track of login failures (incorrect passwords) for each account and can write that information to the GemStone security log (currently, the Stone’s log). By default, messages are logged when the same account fails login attempts 10 or more times within 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 18 Mar 1996 09:39:40 PST   ---
    GemStone user DataCurator has failed on 10 attempt(s)
       to log in within 1 minute(s).
    The last attempt was from user account writer1 on host
       name docs.
```

## Disabling Further Login Attempts

If login failures continue, the Stone repository monitor can disable the account by changing the GemStone password to an invalid one (that is, to a password that cannot be entered). 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, which is also appended to the GemStone security log (currently, the Stone’s log):

```
Login failed: the userProfile has an invalid password.  See
your system administrator to get a valid password.
```

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 under "To Change Another User's Password" on pages 6-12 and 6-24.

## To Find Out Which Accounts Have Been Disabled

Privileges required: OtherPassword.

The message `AllUsers findDisabledUsers` returns a SortedCollection of UserProfiles that are disabled by one of the security precautions 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 ga2
  #2 ga3
```

In each case, the account has been disabled by setting its password to one that is invalid. DataCurator or another user with the OtherPassword privilege can reactivate an account by giving it a new password. For information about how to do that, see pages 6-12 and 6-23.

## To Verify That an Account Is Disabled

Privileges required: OtherPassword.

You can verify that a particular account is disabled by sending the message `isDisabled` to the account's UserProfile. The method returns either True or False. This example inquires about account *qa2*:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') isDisabled
%
true
```

## To Find Out Why an Account Was Disabled

Privileges required: OtherPassword.

You can find out why a particular account was disabled by sending the message `reasonForDisabledAccount` to the account's UserProfile. This example inquires about account *qa2*:

```
topaz 1> printit
(AllUsers userWithId: 'qa2') reasonForDisabledAccount
%
LoginsWithSamePassword
```

The value returned is one of these Strings: 'PasswordAgeLimit', 'StaleAccount', 'LoginsWithSamePassword', or 'LoginsWithInvalidPassword'.

Chapter 7

# Managing the Repository

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 1-43.

This chapter 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 reduce the size of the repository by using GemStone's garbage collection facilities,
- how to audit the repository for consistency and how to identify particularly large objects,
- how to create more space by adding an extent (and optional extent replicate) while the repository is in use, and
- how to replace a corrupted extent with the extent replicate.

# 7.1 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 KBytes). 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
%
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 = /user2/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 10-18). To obtain the free space, multiply FreePages by 8192.

## 7.2 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 on page 7-31.

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.

**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.

# 7.3 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 1-18.

### 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* creates a new repository extent with the given extent file name (specified as a String). The new extent has no maximum size. 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 28 May
1996 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.

## 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 MBytes.

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 7-3.*

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.

# 7.4 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 9-3.

**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 9-7.

## 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 7-3.*

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.

# 7.5 How To Reallocate Existing Objects Among Extents

If you want to reallocate existing objects among two or more extents, the procedure depends in part on whether you are also changing the number of extents. Because changes to DBF_ALLOCATION_MODE directly affect only the subsequent allocation of pages for new or modified objects, additional steps are necessary.

## To Reallocate Objects Among a Different Number of Extents

If you are increasing or decreasing the number of extents and want to change allocation of existing objects as part of that operation, perform a 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 MBytes and you want to divide them about equally between the existing extent and a new one. To populate each extent with about 400 MBytes, follow this procedure:

**Step 1.** Back up your repository using the GemStone full backup procedure described on page 9-3.

**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 1-21). For example:

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

**Step 5.** Restore the repository from your full backup using the GemStone restore procedure described on page 9-7. 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 MBytes and you want to divide them about equally between two existing extents. To populate each extent with about 400 MBytes, follow this procedure:

**Step 1.** Back up your repository using the GemStone full backup procedure described on page 9-3.

**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 MByte repository to remain there, set the size of that extent to 400 MBytes. 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 1-21). 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 9-7. 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*.

# 7.6 How to Manage the Size of the Repository

This section tells how you can use Smalltalk methods to control the size of your repository file. If your system repository has grown large enough to become a problem, first use the methods described in this section.

If you still feel you need to do more, use the methods in the section "How to Remove References to Large Objects" on page 7-36 to test for the presence of objects that exceed a certain size, or for any lingering references in the repository that prevent you from deleting objects and thus freeing up the objects' disk space.

## Overview

It's useful to think of the repository in terms of three levels: pages, objects, and object tables.

- *Page*s represent the physical disk space that contains the repository. Each repository page is 8 KBytes and initially is associated with a particular transaction. Once the transaction is committed, GemStone does not write to those pages until an entire page is no longer needed and has been reclaimed.
- *Object*s are all the objects in the repository. Some of these objects constitute the current committed state of the repository. Others represent old states that are still visible to ongoing transactions. Still others are no longer referenced or visible (commonly called "dead" objects); these objects are ready to be removed so that their identifier and space can be reclaimed.
- *Object tables* represent successive states of the repository by reference to those objects that constituted its committed state at that time. Each session's transaction view of the repository is based on the object table that was current when the transaction started, and that view continues until the session commits or aborts. Consequently, GemStone maintains the objects referenced by that object table even though they may have been dereferenced by subsequent transactions. When the oldest object table is no longer the basis for some session's view of the repository, that table can be reclaimed, as can any objects to which only it refers.

### Why the Repository Grows

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 one MByte:

- 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 KBytes (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 one-half MByte 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 one MByte). If free space falls to that threshold, the Stone repository monitor enlarges the repository.

## GemStone Garbage Collection

As a result of ongoing operations, the repository eventually becomes swollen by a mixture of unreferenced or outdated objects (sometimes called *dead* objects). Removing these objects is a two-phase process:

1. The unneeded objects must be identified, or *marked*, for subsequent garbage collection. GemStone's *epoch garbage collection* feature attempts to mark many of these objects automatically. To mark the others, the GemStone administrator should periodically invoke two methods: `Repository | markGcCandidates` and `Repository | markForCollection`. All three techniques are described later under "To Mark Disconnected Objects" on page 7-15.
2. The actual garbage collection is carried out separately as a *reclaim task*. However, the reclaim task cannot proceed while some ongoing session could still reference those objects in the session's transaction view of the repository. Once all sessions have committed or aborted transactions that were concurrent with the marking phase, garbage collection can proceed. If yours is the only user session logged in and you have the GarbageCollection privilege, you can trigger reclamation by invoking `Repository | reclaimAll`. The reclaim task is described under "To Reclaim Pages" on page 7-20.

For a more extensive overview of GemStone's garbage collection process, refer to Chapter 11.

## GcUser

GemStone provides a special predefined user, GcUser, to run the epoch garbage collection and reclaim tasks. 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 that runs as a child process of the repository monitor. Parameters to control the epoch and reclaim tasks are stored in GcUser's UserProfile. 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.

One GcUser parameter is of general interest, in addition to those discussed later that are specific to the epoch GC and reclaim tasks:

`#GemIOLimit` limits the I/O rate of the Gem through which GcUser logs in (that is, the GcGem). The setting is in operations per second and must be ≥ 1; the default is 5000 per second. For general information about the I/O limit, see "To Tune Garbage Collection of the NotConnectedSet" on page 2-12.

To modify the above garbage collection parameter or similar ones described later, log in as GcUser and send the message `at:`*aKey* `put:` *aValue* to UserGlobals. For instance, to set `#GemIOLimit` to 100, do this:

```
topaz> set user GcUser password thePassword
login
topaz 1>  run
UserGlobals at: #GemIOLimit put: 100 .
System commitTransaction
%
```

Changes to #GemIOLimit are not detected in a timely manner when the GcGem is executing a long-running primitive, such as the one invoked by `markForCollection`. To change the I/O rate during execution of such primitives, use the procedure described under "Changing the I/O Limit During a Long Primitive" on page 2-15.

The following instance methods in `Repository` shut down the GC session for their duration: `auditWithLimit`, `repairWithLimit`, `objectAudit`, and `shrinkExtents`. The method `Repository | restoreFromBackup:` shuts down the GC session when it begins, and if it succeeds, the GC session remains shut down until `restoreFromCurrentLogs` has been successfully executed.

# To Mark Disconnected Objects

Effective removal of dead objects from the repository begins by identifying those objects in one of three ways: epoch garbage collection, the method `markGcCandidates`, or the method `markForCollection`. The following topics describe the data curator's role in each technique.

Remember that the marking of dead objects is only the first part of the garbage collection process. The second part, the reclaim task, runs automatically once the marking phase has been completed *and other sessions have either committed or aborted transactions that were underway*. Until other sessions have acted, marked objects are only *possibly* dead. For a discussion of the reclaim task, see "To Reclaim Pages" on page 7-20. For background about garbage collection, see Chapter 11.

## To Tune Epoch Garbage Collection

The epoch garbage collection (GC) task ordinarily runs automatically and marks objects that are both created and dereferenced within the same set of transactions. This type of garbage collection is most effective in applications where many small transactions update a few previously committed objects.

You may need to adjust the epoch GC parameters so that the time length of an epoch is greater than the average lifetime of the objects you want to detect. The longer the length of the epoch in relation to the average lifetime of your short-lived objects, the greater the percentage of dead objects that will be detected and marked as disconnected at the end of an epoch.

GcUser runs the epoch garbage collection process as needed according to these parameters stored in GcUser's UserGlobals:

| | |
|---|---|
| `#epochGcEnabled` | the default is `true` |
| `#epochGcTimeLimit` | minimum interval in seconds between epochs, must be ≥ 5; the default is 900 seconds (15 minutes) |
| `#epochGcTransLimit` | transaction count threshold to trigger an epoch; the default is 5000 transactions |
| `#epochGcByteLimit` | bytes-written threshold to trigger an epoch; the default is 5000000 bytes |

Each time the Stone processes a commit, it increments the transaction count and updates a count of bytes written during transactions. GcUser triggers a new epoch when the elapsed time since the last epoch exceeds `#epochGcTimeLimit` and either the transaction count during the epoch exceeds `#epochGcTransLimit` or the byte count during the epoch exceeds `#epochGcByteLimit`.

Short epochs may be unproductive because epoch garbage collection detects only those objects that are created and die in the same epoch. On the other hand, lengthy epochs may incur excessive I/O to read all objects in the write set. For a discussion of epoch length, see "Performance Factors" on page 11-23.

The parameter `#deferEpochReclaimThreshold` causes the GcGem to defer epoch garbage collection while the backlog of pages needing reclamation exceeds that threshold. See the parameter description on page 7-23.

The parameter `#epochGcStatsEnabled` enables the collection of statistics, which are then stored in the repository. The default is false, in which case `#epochGcStats` is an empty array. When statistics are enabled, each time epoch garbage collection runs, it adds a four-element array to `#epochGcStats`:

#1 the dateTime that the epoch garbage collection completed
#2 the number of live objects scanned in that run
#3 the number of possible dead objects found in that run
#4 the total number of bytes contained in dead objects found in that run

These statistics should be examined for how often epoch garbage collection is occurring and how many objects it is detecting. In some cases, you might tune `#epochGcTimeLimit` to be quite large, on the order of several hours or a day. For an example of how to change the parameters, see page 7-14.

## To Run markGcCandidates

Privileges required: GarbageCollection.

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 considered a *weak reference*, meaning that it does not prevent collection of the candidates by `markForCollection`.

Because `markGcCandidates` depends on the application providing a list of objects as hints to be swept for references, it is useful only with a cooperating

application. You can check the queue by sending it 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 empties GcCandidates into a temporary array and 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.

When the method completes successfully, it returns an array containing the number of possible 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.
```

## To Run markForCollection

Privileges required: GarbageCollection.

Some objects will escape epoch garbage collection and may not have been added to the GcCandidates queue. The method `Repository | markForCollection` marks any unreferenced objects within the entire repository. This method finds unreferenced objects that existed outside of garbage collection epochs or that were created in one epoch and dereferenced in another. A user with the GarbageCollection privilege can invoke this method occasionally.

Be aware that markForCollection itself may make the repository larger. The repository grows in proportion to the total number of objects (live or dead) in the repository, adding ((numberLive + numberDead) / 4) bytes.

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, you should consider the steps described under "Reducing Impact on Other Sessions" on page 7-19.

When markForCollection completes successfully, this message appears on the screen:

```
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.

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, you should consider temporarily raising the #StnSignalAbortCrBacklog internal parameter if necessary to let markForCollection run for about five minutes between such signals; the necessary value will depend on the commit rate for your application. (Only SystemUser can change the #StnSignalAbortCrBacklog parameter.) For information about setting internal parameters, see "To Change Settings at Run Time" on page 1-39.

If another garbage collection (epoch or markGcCandidates) is in progress at the time you try to do markForCollection, it may report a conflict error similar to that shown below. Try markForCollection again after a few minutes.

```
The Garbage Collection process detected a concurrency
conflict, reason:
#Garbage collection in progress by another session, try
again later.
```

## Performance Characteristics

Tests of `markForCollection` currently show elapsed times ranging from about 3 to 7 seconds per MByte for repositories up of to 10 GBytes. These data were collected on a lightly loaded SPARCserver 1000 with no other users logged in to GemStone. The actual time for running `markForCollection` on a particular repository will depend on the distribution of objects in the repository, the degree of connectivity between the objects, and other system activity. System I/O capacity can impose an additional limitation on performance. The same tests showed I/O activity ranging from about 50 to 175 I/Os per MByte in the repository.

## Reducing Impact on Other Sessions

Because `markForCollection` can make extensive use of system resources for a lengthy period, you may need to reduce its impact on other sessions. Consider the following actions:

- 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). Setting of the parameters is described in "To Tune Garbage Collection of the NotConnectedSet" on page 2-12.
- 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 65536 KBytes.
- 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.

### Scheduling markForCollection

To invoke `markForCollection` through the **`cron`** facility, create a three-line script file similar to the Topaz example on page 7-18 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 Reclaim Pages

When the STN_GC_SESSION_ENABLED configuration option is set to True, the GcUser session automatically performs the reclaim task while user sessions are running. The reclaim task examines pages marked reclaimable because they contain either obsolete views of the repository or objects marked for garbage collection. It also reclaims fragments of space left by transactions that did not fill an entire page.

The reclaimed space will not appear as free space in the repository until other sessions have committed or aborted the transactions that were concurrent with the transaction that reclaimed the objects. There are two steps the data curator can take:

- If your session is the only one logged in (other than GcUser), you can trigger the reclaim task. The procedure to do that is described next.
- If other users are logged in, you can determine which sessions are blocking the reclaim task because their transaction view of the repository currently is the oldest. Reclamation cannot proceed until those sessions either commit or abort their current transaction. See “To Identify Sessions Holding Up Page Reclamation” on page 7-21.

## To Force Reclamation

Privileges required: GarbageCollection.

Reclaiming of objects previously marked as dead can be done explicitly by invoking `Repository | reclaimAll`. This method must be invoked while you are the only user logged in. Follow this procedure:

**Step 1.** Make sure you are the only user logged in (other than GcUser). See "How to Enter Single-User Mode" on page 7-3.

**Step 2.** Send the message `reclaimAll` to the repository. For example:

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

Although reclaim processing will have completed when the method returns true, the space may not show up as free pages in the repository until after the next checkpoint.

Be sure to check the return value. False indicates that reclamation did not succeed because another user logged in.

This method logs out GcUser and suspends all logins until it completes.

## To Identify Sessions Holding Up Page Reclamation

Privileges required: To access information about another session, SessionAccess.

Reclaiming of 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 it's helpful to identify which sessions are holding on to the oldest commit record to understand why reclamation is not proceeding. The method `System | sessionsReferencingOldestCr` returns an array of session Ids, which can be mapped to GemStone logins through `System (C)| currentSessionNames` or `System (C)| 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 *GemStone Kernel Reference*.

## To Tune Parameters for the Reclaim Task

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 should sleep if there is no work scheduled, must be ≥ 3; the default is 10 seconds

`#sleepTimeBetweenReclaim` — the minimum amount of time in seconds that the process should sleep between reclaims even when there is work scheduled; the default is 0 seconds

`#reclaimMinPages` — the minimum number of pages to process in a single reclaim (the reclaim task does not run 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

Three additional parameters affect the reclaim task by batching its input or by deferring other garbage collection activity when the backlog of pages awaiting reclamation exceeds the threshold they set:

`#promoteDeadLimit` — limits the number of previously marked objects that GcUser can declare dead during a single run. This

parameter provides a way of partitioning the work load for the reclaim task. The default limit is 1000 objects.

`#deferPromoteDeadReclaimThreshold`
causes the GcGem to defer declaring more objects dead while the reclamation backlog exceeds its value. Declaring more objects dead adds to the backlog, and a large backlog increases the overhead for other sessions. The default threshold is 1500 pages.

`#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 7-14.

The parameter `#reclaimStatsEnabled` enables the collection of reclamation statistics, which are then stored in the repository. The default is `false`, in which case `#reclaimStats` is an empty array. When the statistics are enabled, each time the reclaim task runs it adds a three-element array to `#reclaimStats`:

#1 the dateTime that the reclamation completed
#2 the number of pages read in that run
#3 the number of pages reclaimed in that run

These statistics should be examined to see how often the reclaim task is running and how many pages are being reclaimed.

The method `System | 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 “To Monitor Page Reads and Writes by a Session” on page 10-7.

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 adding objects, and hence pages, to the reclaim stream by reducing the `#deferPromoteDeadReclaimThreshold` and to defer epoch garbage collection by reducing the `#deferEpochReclaimThreshold` . These changes allow more of the GcGem’s time to be devoted to the reclaim task.

## To Check Page Fragmentation

Space within the repository is managed in pages having a fixed size of 8 KBytes. 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.

## To Shrink Extents

There are two ways to shrink the repository files. The first way, which can be done while the repository is on line, removes unused pages at the end of each extent. While that method may be adequate to meet temporary space needs, it does not compact the extent — it only removes any pages between the last page in use (the *high water mark*) and the end of the extent.

The most effective way to shrink extents requires taking the repository off line and restoring it from a backup. Because the restore method compacts the extent, it results in greater shrinkage. Under ordinary operation, this procedure is seldom

worthwhile because the factors that caused the repository expand in the first place (such as garbage and headroom for sessions logging in) are likely to be repeated.

## To Shrink Extents On Line

Privileges required: FileControl and GarbageCollection.

You may be able to free space in the repository and shrink the extent files while the repository is on line. This procedure is especially useful in temporarily reducing the size of files expanded by the DBF_PRE_GROW configuration option. GemStone returns the file space to the operating system for other uses. This procedure does not affect extents on raw disk partitions.

*NOTE*

*This procedure does NOT remove free space between the beginning of the extent and the high water mark, which is the last active page. If the current high water mark is at the end of the extent, valid objects on that page keep the repository from decreasing in size.*

**Step 1.** Send your repository the message `shrinkExtents`. For example:

```
topaz 1> run
SystemRepository shrinkExtents
%
```

This method truncates extents where possible by removing free space between the last active page in each extent and the end of the file containing the extent. If the last active page (the high water mark) is the last one in the extent, this message has no effect.

**Step 2.** Check to see if sufficient space is free. It you need to free more space, continue with the following steps.

**Step 3.** Mark your repository for garbage collection. For example:

```
topaz 1> run
SystemRepository markForCollection
%
```

For further information about this method, see “To Mark Disconnected Objects” on page 7-15.

**Step 4.** 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.

- If you are the only user logged in to GemStone, send the message **`System reclaimAll`**, which triggers reclamation of the dead objects. When it returns True, reclamation is complete.
- If other users are logged in, the time required also depends 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 7-21.

**Step 5.** Send your repository the message `shrinkExtents` again as in Step 1.

If the DBF_PRE_GROW configuration option is set to True, the extents will be grown again the next time the Stone is started.

## To Shrink Extents Off Line

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 "To Mark Disconnected Objects" on page 7-15.

**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.

- If you are the only user logged in to GemStone, send the message **`System reclaimAll`**, which triggers reclamation of the dead objects. When it returns True, reclamation is complete.
- If other users are logged in, the time required also depends 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 7-21.

**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 9-5.

**Step 4.** Take the repository off-line:

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

**Step 5.** Remove the existing repository extents. Also remove any extent replicates. Obtain a copy of the distribution repository as the first (primary) extent by using the **copydbf** command. For example:

```
% cd $GEMSTONE/data
% rm extentNames
% copydbf $GEMSTONE/bin/extent0.dbf primaryExtentName
```

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

**Step 6.** To put the repository back on line, issue the **startstone** command:

```
% startstone gemStoneName
```

If you do not specify *gemStoneName*, **startstone** defaults to `gemserver50`.

**Step 7.** Log in to linked topaz again.

*NOTE*

*To perform the remaining parts of this procedure, you must be the only person logged in to GemStone. Logins will be disabled when you start the next step.*

**Step 8.** Restore the repository by using the method `Repository | restoreFromBackup:` *fileOrDevice*, using the same file or device as in Step 3. Because it is being restored into a copy of the initial repository, the restored repository will be compressed to the minimum space. For example:

```
topaz 1> run
SystemRepository restoreFromBackup: '/users/bk/oct31'
%
```

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 9-12.

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 9.** If the repository was in full transaction logging mode (that is, STN_TRAN_LOGGING_ENABLED 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 10.** Enable logins again:

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

# 7.7 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 only to ensure page-level consistency after some kind of system failure, such as a read-write error or a cache coherency error. In these cases, a successful page audit indicates that the problem did not affect the committed repository. GemStone must be halted when you perform a page audit.
- An *object audit* checks the consistency of the repository at the object level and generates useful statistics in the process. An object audit can be performed as part of routine maintenance and is always performed while GemStone is running.

## To Perform a Page Audit

Page audits allow you to diagnose problems in the system repository by checking for consistency at the page level. You do not need to run this utility as part of routine maintenance of the repository.

The **pageaudit** utility can be run only on a repository that is not in use.

To check for page-level problems, run **pageaudit** on the repository defined in your ordinary GemStone configuration by issuing this command at the operating system level:

`% pageaudit [ gemStoneName] [ -zsystemConfig] [ -eexeConfig] [-h]`

where

*gemStoneName* is the name of the GemStone repository monitor,

*systemConfig* is the system configuration file, and

*exeConfig* is the executable configuration file.

All four arguments are optional in a standard GemStone configuration. If these options are not supplied, **pageaudit** uses `gemserver50` for *gemStoneName*.

See Appendix B for more information about the **pageaudit** command; or, for on-line documentation, type **`pageaudit -h`** or `man pageaudit` .

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

```
PAGE AUDIT STATISTICS  mozart sun4m (Solaris 2.5 Generic)
- Wed 01 May 1996 09:53:46 PDT

8192 bytes = 1 GemStone Page = 1 disk blocks
1048576 bytes = 1 Mbytes
Repository Size                                  8 Mbytes
Data Pages                                       5 Mbytes
     Page headers for data pages                 0 Mbytes
     Free space in data pages                    0 Mbytes
Meta Information Pages                           0 Mbytes
Shadow Pages                                     0 Mbytes
Free Pages                                       2 Mbytes
**** Number of differences found in page allocation = 0.

Page Audit of logical repository completed successfully.
```

The report contains the following statistics:

Repository Size — The repository size is total physical size, the same size the operating system reports for an extent file.

Data Pages — The data pages are all pages referenced from the object table. The breakdown shows the space occupied by page headers on data pages and the total size of unused space fragments. GemStone tries to limit the unused space in each page to a small percentage, currently 10 percent. Whether that is successful depends in part on the size of typical objects in the repository, the worst case being a large number of objects, each of which is slightly larger than half of an 8 KByte page.

Meta Information Pages — Meta information pages are pages that contain only internal information about the repository, such as the object table.

Shadow Pages — Shadow pages are pages scheduled for scavenging by the reclaim task.

Free Pages — Free space in the repository is computed as the number of free pages times the size of a page (8 KBytes). 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 9-7.)

### Performance Characteristics

Tests of **pageaudit** currently show elapsed times of about 2 seconds per MByte for repositories up to 10 GBytes. These data were collected on a lightly loaded SPARCserver 1000.

## To Perform an Object Audit and Repair

Privileges required: GarbageCollection.

Object audits check the consistency of the repository at the object level. The level of operation depends on whether the repository monitor is in single-user mode:

- If your session is the only one logged in, then logins are disabled for the duration of the audit, page reclamation is forced to complete, and additional consistency checks are made. You can have a higher degree of confidence in a successful object audit that is performed in single-user mode. The reclamation effect is the same as that of `Repository | reclaimAll .`
- If other users are logged in, the object reclamation is not completed prior to the audit. As a result, less checking can be performed, and the degree of confidence is reduced.

The GcGem is automatically shut down for the duration of the audit.

GemStone provides two methods for auditing objects. `Repository | objectAudit` is the simplest to use. It reports all errors it encounters, but statistics are reported only for objects larger than 100000 bytes or Oops. If you want statistics with a different reporting threshold, use the method `Repository | auditWithLimit:` *sizeLimit*.

These methods abort the current transaction, and the garbage collector session is shut down for their duration.

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

**Step 2.** If Stone has been restarted following a system crash (rather than an orderly shutdown) or if the repository has been restored from a backup, always invoke `markForCollection` before performing an audit:

```
topaz 1> run
SystemRepository markForCollection
%
```

Failure to perform `markForCollection` before the audit under these circumstances can result in errors being reported even though the repository is not corrupted. Such error reports result from unfinished garbage collection activity at the time of the backup or the crash.

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

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

The object audit begins with an optimized scan of the Object Table and the data pages. This scan provides the quickest possible result if no errors are detected. 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 7-34. If errors are detected, GemStone ordinarily re-scans the repository to provide detailed information.

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 the re-scan 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 7-3. 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.

## Performance Characteristics

Tests of `objectAudit` currently show elapsed times ranging from about 2 to 4 seconds per MByte for repositories up to 10 GBytes. These data were collected on a lightly loaded SPARCserver 1000 with no other sessions logged in to GemStone.

## Understanding Object Audit Statistics

Figure 7.1 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.

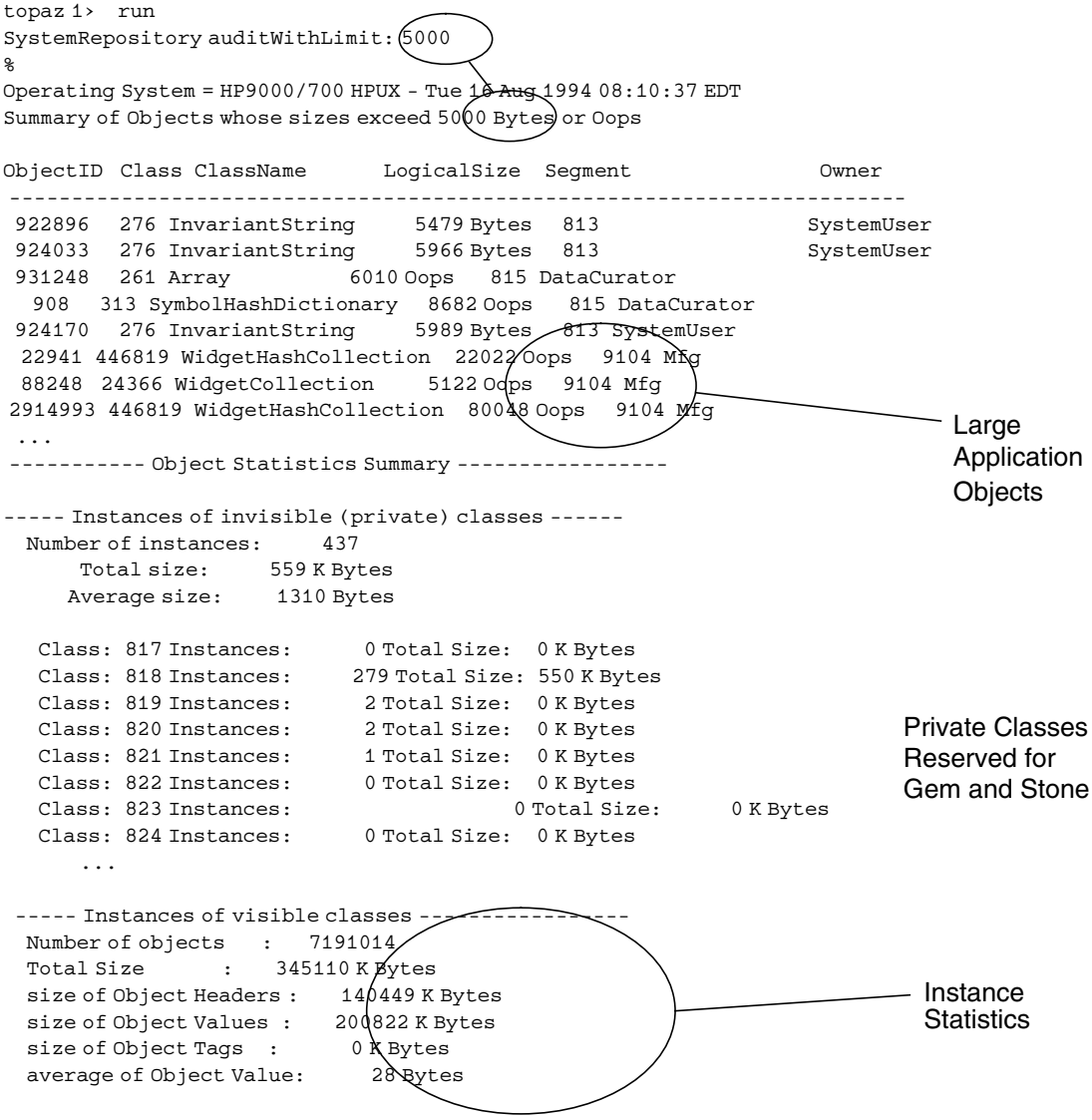
2. Statistics about invisible (private) classes that are reserved for GemStone's use. The number of these classes varies from release to release, and some may not be used in a particular release.

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 1-18). 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 `Object | tagAt:` and `tagAt:put:` in the *GemStone Kernel Reference*.

**Figure 7.1 Statistics From an Object Audit**




```
topaz 1> run
SystemRepository auditWithLimit: 5000
%
Operating System = HP9000/700 HPUX - Tue 16 Aug 1994 08:10:37 EDT
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  80018 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: Successful Completion, no errors detected.
Completed execution of object audit.  0 objects contained errors.
topaz 1>
```

# 7.8 How to Remove References to Large Objects

One way to obtain free space in the repository is to remove large objects that are no longer needed. Removing the objects involves three steps: identifying the objects themselves, finding all references to them (which keep the objects from being garbage collected), and then removing those references.

## To Identify Large Objects in the Repository

Before you execute any of the following expressions, first make sure that the Topaz display level is sufficient to display the desired information. Use the Topaz **level** command to raise the level to at least 1:

```
topaz 1>  level 1
```

The following expression causes GemStone to look through the symbol list for each user in AllUsers and gather information on any named objects larger than the SmallInteger *aSize*:

```
topaz 1>  run
AllUsers findObjectsLargerThan: aSize limit: aSmallInt
%
```

The information is returned as an Array of up to *aSmallInt* elements. Each element is 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 the System object instead:

```
topaz 1>  run
System findObjectsLargerThan: aSize limit: aSmallInt
%
```

*NOTE*

*This method may take a considerable length of time to complete.*

In this case, GemStone returns an Array of all objects in the repository larger than the SmallInteger *aSize*—whether or not they are named in a user's symbol list. 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
%
```

This method returns an Array of objects in the repository that reference *anObject*. If an object contains multiple references to *anObject*, that object will be present 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 a considerable length of time to complete. In addition, the resulting Array may consume a large amount of disk space.*

To limit the cost in disk space required for the result, you can 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`, which allows the object to be removed during the normal garbage collection process outlined in “How to Manage the Size of the Repository” on page 7-12. De-referencing objects must be done through a Smalltalk program.

## 7.9 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:

```
--- 6 May 1994 17:04:28 Pacific Daylight Time
    Log message from user = DATACURATOR
    Repository Read failure
    Filename = ... /data/extent1.dbf
    PageID = 559
Filename = ... /data/extent1.dbf
```

**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 9-7.

**Step 5.** Restart the Stone repository monitor.

# 7.10 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 7-29 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 9-7.)

## 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 7-29. 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 9-7). 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 7-29 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 9-7).

*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.*

# 7.11 How to Perform Bulk Loading

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 KBytes 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 2-9.
- 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.)

Chapter

# 8 Managing Transaction Logs

## 8.1 Overview

A transaction log contains the information necessary to re-do 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. 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 8.1.) Collectively, the log files logically form an almost infinite sequential file with a maximum size of $4 \times 10^6$ 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 8.1 compares full and partial transaction logging.

**Table 8.1 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.<br>The maximum number of file system logs on line 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.<br>The maximum number of logs on line at one time depends on the number of directories or raw partitions in the list. |
| Action when log space becomes full | The Stone pauses execution if it cannot find space in any of the specified directories or raw partitions. | The Stone deletes log files from the circular list of directories and keeps running. |
| Administrative task | Monitor log space; archive log files and delete them as necessary | None |

## Use in Recovery from an Unexpected Shutdown

Between checkpoints, GemStone writes each committed transaction to a transaction log *(*Figure 8.1*)*. Then, in the event of a system crash, GemStone can recover by automatically reapplying transactions from the log to the latest checkpoint (Figure 8.2).

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 UNIX methods 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 8.1 System Time Line: Normal Operation**

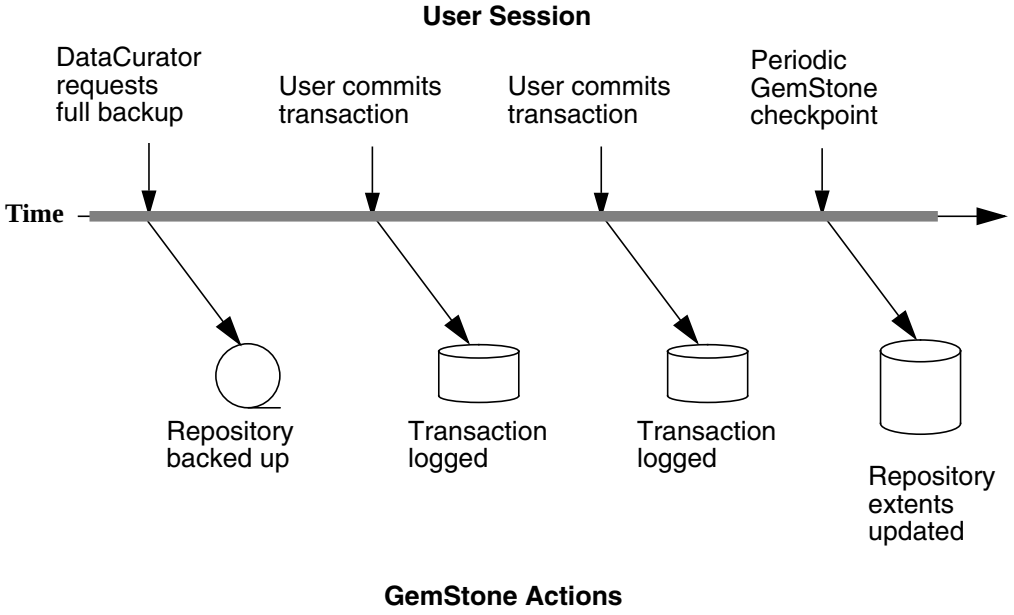
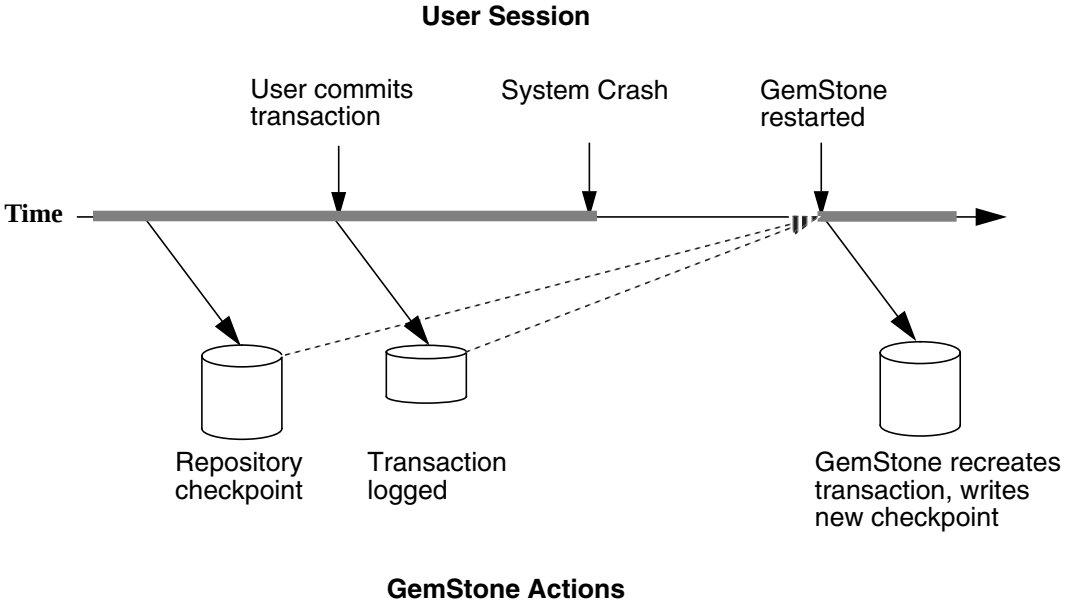


**Figure 8.2 System Time Line: System Crash**




## 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 9-3.

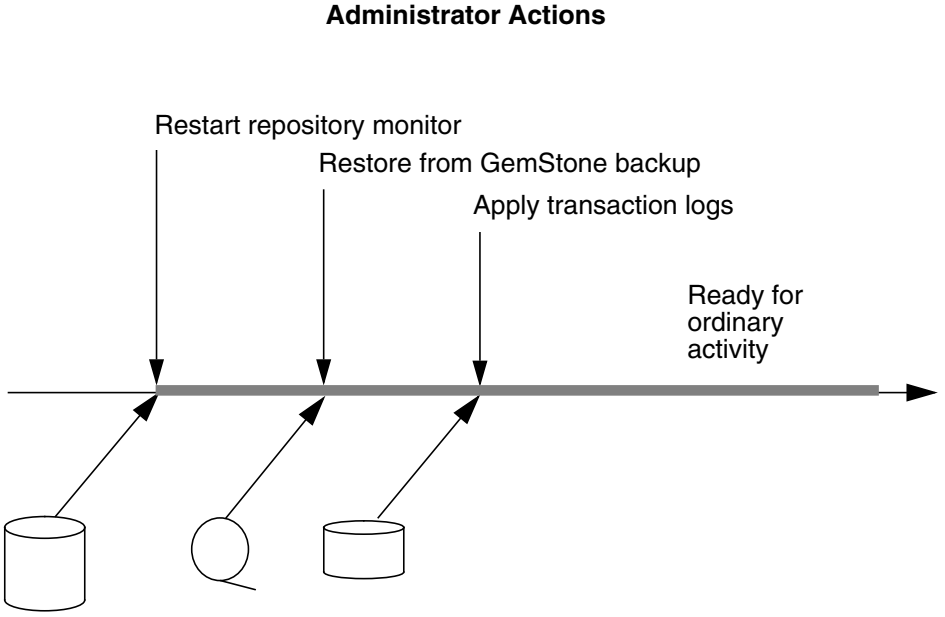
## How the Logs Are Used

The GemStone restore procedure (Figure 8.3) 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 9-13.)

*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 8.3 System Time Line: Restoring a GemStone Backup**



# 8.2 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 on line.
- 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 UNIX tools, such as **tar** and **cp**, can be used to move log files to other locations or to archival media. We recommend that you archive and free 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 8.2.

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.

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 17 Jun 1996 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 17 Jun 1996
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/archivelog.sh`. You will need to edit

the script to conform to your own partition names and archive location, and then test it.

## To Add a Log 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 (C)| 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 1-38. The Repository methods in Table 8.2 return other information that is helpful in managing transaction logs.

**Table 8.2 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. |
| `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 description in the *GemStone Kernel Reference*. |
| `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 any time by using the method `Repository | startNewLog` . The method 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
```

## 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 9-3.

**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 | restoreFrom:` . See "How to Restore a GemStone Repository" on page 9-7.

# 8.3 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 2 MBytes 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

# 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 provided you comply with the precautions this chapter describes.

## 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.

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.

You should also make backups of transaction logs, especially if you have enabled full transaction logging. Such 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 UNIX utilities. For a related discussion, see "To Archive Logs" on page 8-7.

## Backups Are Made While Gemstone Is Running

GemStone 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.

The backup is made outside of a transaction so it does not interfere with ongoing garbage reclamation activity. 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 Tune Garbage Collection of the NotConnectedSet" on page 2-12.

## 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 9-21.

## 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 on line 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, raw partition, 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 9-5.

*WARNING*

*If* fileOrDevice *runs out of space, such as off the end of a tape, the backup will terminate with a system I/O error at that point. 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. This argument is especially useful when the size of the backup exceeds the capacity of a single tape. For further information and an example, see "To Create a Backup in Multiple Files" on page 9-5. 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.

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/apr25.96'"
SystemRepository fullBackupTo:'/users/backups/apr25.96'
%
true
```

The backup file named `apr25.96` 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`**.

If the backup file already exists, or if a raw disk partition already contains a GemStone extent, transaction log, or backup, the method returns an error. Use the **removedbf** utility to erase raw disk partitions.

If the *fileOrDevice* argument is an empty string, an error will be returned. You must specify the name of a file, raw partition, or device, not a directory name.

## Performance Characteristics

Tests of `fullBackupTo:` currently show elapsed times of about 1.5 seconds per MByte for backups to tape, and 1 second or less per MByte for backups to disk. These data were collected on a lightly loaded SPARCserver 1000 for repositories of up to 10 GBytes. In each case, the devices and the repository were local to the node on which the backup was performed.

## To Create a Backup on a Remote Node

The following example uses an NRS to access a tape drive located on another node. The same approach can be used to access a remote disk. Of course, performing a backup across the network is likely to take longer than writing it to a local device.

```
topaz 1> run
"Access a tape device on node flute"
SystemRepository fullBackupTo:'!@flute!/dev/rst0'
%
```

A GemStone NetLDI must be running on the remote node. The user performing the backup must provide authentication for that node, such as an entry in a `.netrc` file. The requirements are similar to those given in Chapter 3 for starting an RPC Gem session process on a remote node.

## To Create a Backup in Multiple Files

To create a multiple-file backup, include the argument `MBytes:` *byteLimit*. For instance, `MBytes:10` tells GemStone to limit the size of the backup file to 10 MBytes. When backing up to tape, use the tape's available megabytes as the `MBytes:` argument.

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
      '/usersbackups/apr25.95'"
SystemRepository fullBackupTo: '/users/backups/apr25.95'
MBytes: 10
%
Finished writing backup file 0 of a multifile backup
```

To create the next file in the backup, send your repository the message `continueFullBackupTo:` *fileOrDevice* `MBytes:` *mByteLimit*.

For example:

```
topaz 1> run
 "Continue a full backup by writing a second file to
      '/users/backups/apr25.95_2'"
SystemRepository continueFullBackupTo:
'/users/backups/apr25.95_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 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/apr25.96 /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/apr25.96 started at Apr/25/1996
13:22:28
```

# 9.3 How to Restore a GemStone Repository

Privileges required: FileControl.

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.

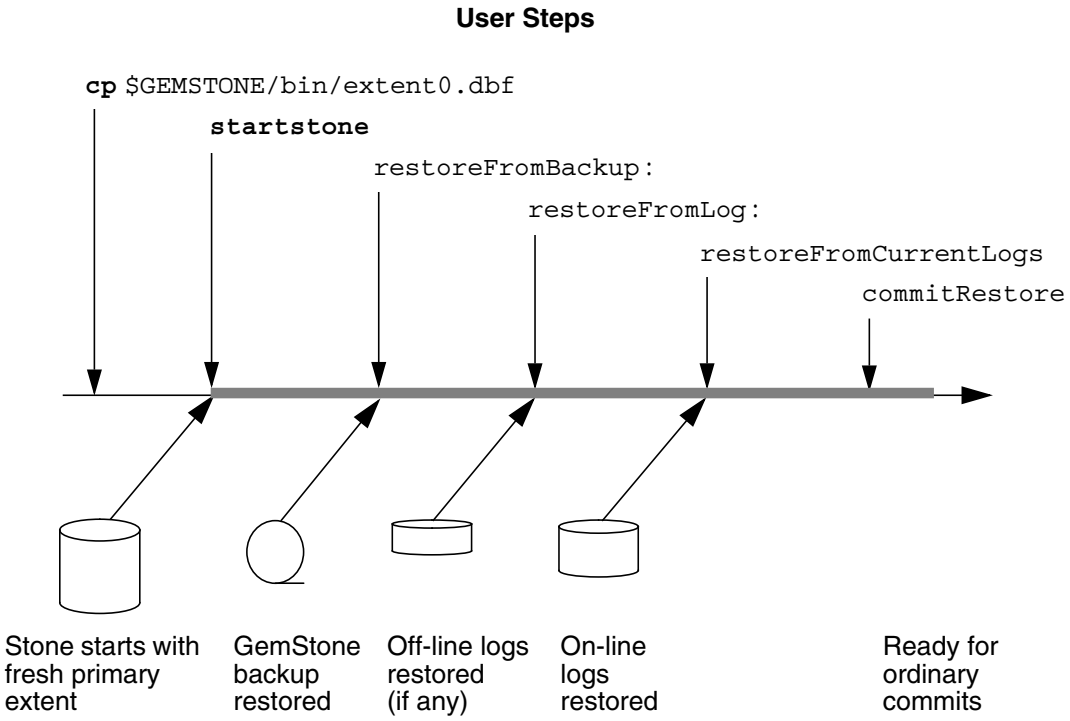
To protect your data, we recommend that you restore from backups created by GemStone, not from backups created by operating system commands such as **dump**, **tar** or **cp**. GemStone backups are created by the method `Repository | fullBackupTo:`. These backups are the most reliable because they are 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 9-21.)

*NOTE*

*Backups created by a dump of the whole file system may or may not be usable. Before you consider restoring from UNIX-level backups, be sure to read "How to Restore from an Operating System Backup" on page 9-21.*

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 9-3.) If full backups of your repository require more than one tape, restoring tape 1 without tape 2 does not give you a usable version of the repository. The objects on tape 1 are copied, but the transaction cannot be committed until you restore tape 2. (The Topaz **commit** command does not commit a partial restore, even though the message may say the commit was successful.)

**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, such clustering in the original repository takes precedence over the DBF_ALLOCATION_MODE configuration setting used by the Stone performing the restore operation. If the number of extents differs, then the DBF_ALLOCATION_MODE setting at the time of the restore controls the distribution of objects across extents.

## 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, especially if you are copying to or from a raw partition.

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.

*WARNING*

*Do NOT delete the transaction log files as of the crash—leave them on line without moving them.*

**Step A4.** Remove all extent files specified in DBF_EXTENT_NAMES in your configuration file. Remove all replicates specified in DBF_REPLICATE_NAMES.

**Step A5.** Copy the distribution extent to the location of your primary extent, which is the extent listed first in DBF_EXTENT_NAMES. For example:

```
% copydbf $GEMSTONE/bin/extent0.dbf \
  $GEMSTONE/data/extent0.dbf
% chmod 600 $GEMSTONE/data/extent0.dbf
```

Use **chmod** to give the copy the same permissions you ordinarily assign to your repository files.

**Step 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 when you start the restoration process (Step A9). In addition, you can force closure of the current log and opening of a new log at any time by using the method `Repository | startNewLog`. That method returns a SmallInteger, which is the `fileId` of the new log.

**Step 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*. The argument *fileOrDevice* can be the name of a file, a tape device, or a raw disk partition. For information about using tapes, see “To Restore Backups from Tape” on page 9-12.

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 9-12.

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: '/users/bk/apr25.96'
%
```

If the full backup is contained in one file, the system commits the restore and returns a summary and status. For instance:

```
Restore from full backup completed with 30569 objects
restored and 0 corrupt objects not restored.
```

- The next step depends on the setting of the STN_TRAN_FULL_LOGGING configuration option at the time the backup was made. 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.

- When full logging is enabled, the status line is similar to this:

  ```
  Ready for restore from transaction log(s).
  ```

  Continue with “B. To Restore Subsequent Transactions” on page 9-13.

*CAUTION*

*Although you can end the restore process before restoring from all transaction logs, doing so can make it impossible to restore the omitted logs later by repeating the process. If you plan to terminate the restore prematurely, first read “Precautions When Restoring a Subset of Transaction Logs” on page 9-19.*

## Performance Characteristics

Tests of `restoreFromBackup:` currently show elapsed times of about 1 to 4 seconds per MByte for repositories of up to 10 GBytes. These data were collected on a lightly loaded SPARCserver 1000. These data do not include the time to restore subsequent transactions from transaction logs.

## To Restore Backups from Tape

When restoring from tape, use the device name for the *fileOrDevice* parameter. For example,

```
topaz 1>  run
SystemRepository restoreFromBackup: '/dev/rmt0.4'
%
```

Depending on your operating system and the type of tape drive, you may need to issue operating system commands first. For information, check your operating system documentation.

To access a tape device on another host, make sure a NetLDI is running on that host and then include the host's name in *fileOrDevice* as a network resource string. For instance:

```
topaz 1>  run
SystemRepository restoreFromBackup: '!@flute!/dev/rst0'
%
```

## To Restore Multiple-File Backups

If you are restoring a backup that occupies more than one file or tape, each backup file must be restored in sequence. You can determine the restore status and the internal identification of the backup file required next by using the method `restoreStatus`. If a restore process is active, the reply indicates the date and time to which the repository has been restored, and the type of file and the internal file identification number (`fileId`) of the file that must be restored next. (The `fileId` is reset to 0 at the beginning of each full backup.)

Repeat Step A9 on page 9-10, 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 each file or device in the array in proper sequence. This example restores a backup that occupies two files by placing the file names in an array:

```
topaz 1> run
SystemRepository restoreFromBackups:
#( '/users/backups/May_1.1'
   '/users/backups/May_1.2' )
%
Opening file /users/backups/May_1.1
Opening file /users/backups/May_1.2
Restore from full backup completed with 32804 objects
restored and 0 corrupt objects not restored.
```

When you restore the last backup file, GemStone either commits the restore or indicates that it is ready to restore from transaction logs, as explained on page 9-11.

If an error occurs, the shadow object space being built by the restore reverts to its state as of the end of the last file that was successfully restored.

If you need to cancel the process and start over while restoring a multiple file backup, use the method `Repository | abortRestore`.

If the `Topaz` login session dies before the last backup file has been restored, you must start over by restoring the first file of the set.

## 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. If for some reason you plan to omit one or more log files, first read "Precautions When Restoring a Subset of Transaction Logs" on page 9-19.*

**Step B1.** 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 Thu 21 Dec 1995 14:52:06 PST
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** *fileName* **-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 1-38).

**Step B2.** 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 9-17.

- If 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 B3.
- If several of the older transaction logs have been moved to a different disk location, send `Repository restoreFromArchiveLogDirectories:` ... `replicatePrefix:` to restore those log files. The following example restores logs archived in /GS-archive, which is not one of the active locations listed in STN_TRAN_LOG_DIRECTORIES:

```
topaz 1> run
SystemRepository restoreFromArchiveLogDirectories:
  #( 'GS-archive' )
tranlogPrefix: ''
replicateDirectories: #()
replicatePrefix: '' .
%
```

See the method description in the *GemStone Kernel Reference* for details. A directory location can include an NRS for a remote node, but a NetLDI must be running on that node.

Continue with the next substep if it applies, or else with Step B3.

- If individual transaction logs need to be restored, execute `Repository | restoreFromLog: fileOrDevice` for each file or raw partition in time sequence. (If the files are on tape, the files must first be restored to a disk drive, but they do not need to be in their original location.) For example:

```
topaz 1> run
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to Thu 21 Dec 1995 17:28:13 PST
next fileId = 38
topaz 1> run
SystemRepository restoreFromLog: '/users/tranlog38.dbf'
%
Restore from transaction log succeeded.
```

Continue with Step B3 to restore from on-line logs.

**Step B3.** 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 on line 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 succeeded.
```

**Step B4.** Send the message `commitRestore` , which tells the system you are finished restoring transaction logs.

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

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 28 May 1996 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 28 May 1996 15:33:50 PDT.
  oldest transaction references fileId -1 ( this file ).
  Checkpoint 2 started at: Tue 28 May 1996 15:38:50 PDT.
  oldest transaction references fileId -1 ( this file ).
  Checkpoint 3 started at: Tue 28 May 1996 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 May 28, 1996 at 3:35:00 a.m.:

```
topaz 1>  run
SystemRepository timeToRestoreTo:
 (DateTime fromString: '28/05/1996 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.

## 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 Thu 21 Dec 1995 14:52:06 PST
next fileId = 37,  record = 53
```

Retrieve a copy of the transaction log from a UNIX 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 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 1> run
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to Thu 21 Dec 1995 14:52:06 PST
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 1> run
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to Thu 21 Dec 1995 17:28:13 PST
next fileId = 2
```

The restore status now shows that we are ready for the next transaction log, `tranlog2.dbf`. Because the remaining logs are on line, we will return to our original procedure of restoring them as a group and then commit the restored repository:

```
topaz 1> run
SystemRepository restoreFromCurrentLogs
%
Restore from transaction log succeeded.
topaz 1> run
SystemRepository commitRestore
%
Restore from transaction log 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 9-10) 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 B1, on page 9-13) 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 repository monitor was shut down. That is, the backup must have been started after a **stopstone** and must have been completed before the subsequent **startstone**.

If the file system itself has been corrupted, not just the extent files, see the section "Disk Failure or File System Corruption" on page 4-19.

*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 9-3.*

**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 on line 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.
- 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. Continue with the next step, Step 7.

**Step 7.** Determine the location of all needed transaction logs. The method `restoreStatus` identifies the earliest transaction log that is needed. In this example it is `tranlog37.dbf`:

```
topaz 1> run
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to Thu 21 Dec 1995 14:52:06 PST
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** *fileName* **-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 1-38).

**Step 8.** Restore the transaction logs in time sequence, beginning with the log identified by `restoreStatus`. How you do restore the logs depends on where the oldest logs are located and is described next. If you encounter a failure because of a truncated or corrupted transaction log, refer to "Errors While Restoring Transaction Logs" on page 9-17.

- If 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 9.
- If several of the older transaction logs have been moved to a different disk location, send `Repository | restoreFromArchiveLogDirectories: ... replicatePrefix:` to restore those log files. The following example

restores logs archived in /GS-archive, which is not one of the active locations listed in STN_TRAN_LOG_DIRECTORIES:

```
topaz 1> run
SystemRepository restoreFromArchiveLogDirectories:
  #( 'GS-archive' )
tranlogPrefix: ''
replicateDirectories: #()
replicatePrefix: '' .
%
```

See the method description in the *GemStone Kernel Reference* for details. A directory location can include an NRS for a remote node, but a NetLDI must be running on that node.

Continue with the next substep if it applies, or else with Step 9.

- If individual transaction logs need to be restored, execute `Repository | restoreFromLog: fileOrDevice` for each file or raw partition in time sequence. (If the files are on tape, the files must first be restored to a disk drive, but they do not need to be in their original location.) For example:

```
topaz 1> run
SystemRepository restoreStatus
%
Restoring from Transaction Log files,
restored to Thu 21 Dec 1995 17:28:13 PST
next fileId = 38
topaz 1> run
SystemRepository restoreFromLog: '/users/tranlog38.dbf'
%
Restore from transaction log succeeded.
```

Continue with Step 9 to restore from on-line logs.

**Step 9.** After you restore transactions from any off-line log files, restore transactions from the on-line log files using `restoreFromCurrentLogs` . All the remaining log files must be in directories or raw partitions specified in STN_TRAN_LOG_DIRECTORIES.

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

**Step 10.** If restoration from the transaction logs was successful, send the message `commitRestore` to tell the system that you are finished restoring. No further logs can be restored, and normal user commits will be allowed.

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

**Step 11.** Make a new GemStone backup as soon as operational circumstances permit.

Chapter

# 10 Monitoring GemStone

This chapter tells you where to look for the log files that GemStone processes create and how to monitor the performance of the GemStone server and its clients.

If you decide to keep a GemStone session running for occasional use, be careful not to leave it in an active transaction, because that can impede garbage collection activity until you either commit or abort.

## 10.1 How to Use Manual Transaction Mode

For sessions that are not committing changes to the repository, we recommend that monitoring be done in manual transaction mode. For example, to enter manual transaction mode:

```
topaz> printit
System transactionMode: #manualBegin
%
```

Even in manual transaction mode, it is possible to cause a commit record backlog that interferes with garbage collection. It is desirable to issue an abort frequently or to use an application that handles SignalAbort and LostOtRoot errors from the Stone repository monitor.

If you need to commit a change, first begin a transaction manually:

```
topaz> printit
System beginTransaction .
AllUsers addNewUserWithId: #Jane password: 'gemstone' .
System commitTransaction
%
```

After you commit the transaction (or abort it), your session will return to waiting outside of a transaction.

# 10.2 How to Find GemStone Logs

GemStone creates three types of log files: logs for GemStone object server processes, logs for processes related to individual GemStone sessions, and logs for GemStone network server processes (the NetLDIs).

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 server and the GcGem will be in the same location as the corresponding Stone's log.

## GemStone Server Logs

The Stone repository monitor and its child processes each create a log file in a single location. By default, the files are in `$GEMSTONE/data` and have a name beginning with the name of the repository monitor. Table 10.1 shows typical log names for a repository monitor having the default name of `gemserver50`. Log names for child processes also include the process id and one of these suffixes:

| | |
|---|---|
| `pcmon.log` | for the shared page cache monitor log |
| `pgsvraio.log` | for the AIO page server log |
| `gcgem.log` | for the garbage collector log |

**Table 10.1 Representative Log Names for GemStone Server Processes**

| Typical Name | GemStone Process |
|---|---|
| `gemserver50.log` | Stone repository monitor |
| `gemserver506936pcmon.log` | Shared page cache monitor |
| `gemserver506966pgsvraio.log` | AIO page server |
| `gemserver506980gcgem.log` | Garbage collector session |

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 10.1 are placed in the same directory.
2. A path specified by the GEMSTONE_LOG environment variable. If *logFile* is relative (that is, not a complete path preceded by a /), *logFile* is created in a current directory. Logs for the child processes in Table 10.1 are placed in the same directory.
3. `$GEMSTONE/data/` *gemStoneName*`.log`.

## Stone Log

The log for the Stone repository monitor is cumulative across runs. This log is the first one you should check when a GemStone system problem is suspected. In addition to possible warnings and error messages, the log records 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 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 10.1, it is for processId 6936). Check this log if other messages refer to a shared page cache failure. If the cache monitor exits normally, this log is removed.

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 log for the repository monitor's private AIO page server is located in the same directory as the Stone's log. This log is for a specific page server process and is removed if the page server exits normally.

Configurations having an extent on another node also have a log for the Stone's page server on that node. By default, this log is named `pgsvrmain`*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.

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

### GcGem Log

Each time the Stone repository monitor starts a new garbage collection (Gc Gem) session process, a new log is created in the same location as the Stone's 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. If the GcGem exits normally, the current log is removed. When GcUser logs in again, a new log is opened under a name that includes the new processId 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.

## 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 10.1 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 10.2 Typical Logs Supporting Gem Sessions**

| Typical Name | GemStone Process |
|---|---|
| `gemnetobject27853node2.log` | 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 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 3-15).

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* when starting the NetLDI. Each NetLDI you start with the same name appends to one log, so it's a good idea to remove outdated messages occasionally.

By default, the NetLDI log contains only configuration information and error messages. The configuration information reflects the environment at the time the NetLDI was started and the effect of any authentication switches specified as part of the startup command. The following log description for the default configuration may be helpful for comparison:

```
System password authorization is permitted.
Authentication is required only to create processes.
Process creation is permitted through user's HOME directory.
Created processes belong to client's account.
```

The preceding lines map to NetLDI options in this way:

Line 1 Authentication is not restricted to Kerberos (**-k**).

Line 2 Guest mode is not in use (**-g**), but authentication is not required for all NetLDI services (**-s**).

Line 3 Services are not restricted to those listed in `$GEMSTONE/bin/services.dat` (**-n**).

Line 4 Captive accounts are not in use (**-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.

# 10.3 How to Monitor GemStone Performance

As part of your ongoing responsibilities, you may find it useful to monitor performance of the object server. GemStone supports monitoring at two levels: command-line facilities to determine the general status, and Smalltalk messages that you can use to access statistics maintained in the shared page cache.

Statistics about individual session processes may also be helpful. You can obtain information separately about page reads and writes, or you can obtain more detailed cache statistics about that session.

## To List Running Servers

The **gslist** utility lists all Stone repository monitors, shared page cache monitors, and NetLDIs that are running. The **gslist** command by itself checks the locks directory (`/opt/gemstone/locks` ) for entries. The **-v** option causes it to verify that each process is alive and responding. For example:

```
% gslist-v
Status Version  Owner   Started   Type Name
------ -------- -------- ------------ ------ ----
  OK  5.0         gsadmin    Jan 23 11:49 cache    gemserver50@mozart
  OK  5.0         gsadmin    Jan 23 11:49 Stone    gemserver50
  OK  5.0         gsadmin   Jan 23 11:54 Netldi netldi50
```

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.

## To Monitor Page Reads and Writes by a Session

Privileges required: Statistics.

You can obtain information about session I/O by invoking the methods `System (C)| pageReads` and `System (C)| pageWrites` from that session. These methods return the number of reads and writes performed by that session since its start. For example:

```
topaz 1>  printit
System pageReads
%
19
topaz 1>  printit
System pageWrites
%
0
```

## To Monitor Cache Statistics

Two related methods provide a way to analyze performance by examining statistics collected in the shared page cache. The method `System | cacheStatistics:` *aProcessSlot* returns an array of the information described in Table 10.3. The method `cacheStatisticsDescription` , when used with display level 1 in Topaz, prints the description of each slot as shown in the table. For example:

```
topaz 1>  level 1
topaz 1>  printit
System cacheStatistics: 1
%
an Array
 #1 Stone
 #2 12256
 #3 0
 #4 -1
 #5 416
...
topaz 1>  run
System cacheStatisticsDescription
%
an Array
 #1 ProcessName
 #2 ProcessId
 #3 SessionId
 #4 LockedPage
 #5 AttachDelta
...
```

All array elements except the first are Integers. Some are zero unless the slot being examined belongs to the Stone repository monitor or the shared page cache monitor. The offsets shown in Table 10.3 may change in future releases.

The process slots assigned to GemStone server processes can be determined by examining the first two elements returned (the name and process id). The slot assignments for the first four cache slots on the Stone's node currently are

Slot 0 shared page cache monitor
Slot 1 Stone repository monitor
Slot 2 Stone's AIO page server
Slot 3 GcGem (at server startup)

The GcGem initially is in Slot 3, but if it has been shut down and restarted, it may then be in a different slot.

On nodes remote from the Stone, the page server for the Stone that started the cache will be in Slot 1.

Each Gem or page server has a unique number appended to its name (element 1) so that data can be related to the correct process in the case where several transient processes successively occupy the same cache slot.

The information in Table 10.3 can also be obtained for the cache slot of a Gem session process. The output in certain elements, such as the Stone Statistics category, will be zero. You can use the System class method `myCacheProcessSlot` to return the process slot in the shared page cache that corresponds to the calling process. For example,

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

Statistics marked "obsolete" in Table 10.3 always return zero. These slots may be used differently in future releases.

**Table 10.3 Cache Statistics**

| Array Element[a] | Name | Cache Slots for Which Available |
|---|---|---|
| 1 | ProcessName, aString | All |
| 2 | ProcessId | |
| 3 | SessionId | |
| 4 | LockedPage (obsolete) | |
| 5 | AttachDelta | |
| 6 | AttachedCount | |
| 7 | SharedAttached | Shared page cache monitor only |
| 8 | TotalAttached | |
| 9 | FreeFrameCount | |
| 10 | LocalPageCacheHits | All |
| 11 | LocalPageCacheMisses | |
| 12 | LocalPageCacheWrites | |
| 13 | PageReads | |
| 14 | PageWrites | |
| 15 | CommitCount | Gems only |
| 16 | FailedCommitCount | |
| 17 | AbortCount | |
| 18 | ObjsCommitted | |
| 19 | NewObjsCommitted | |
| 20 | TotalObjsCommitted (obsolete) | |
| 21 | TotalNewObjsCommitted (obsolete) | |
| 22 | ScavengeCount | |
| 23 | TimeInScavenges | |
| 24 | DeadObjCount | |
| 25 | MakeRoomInOldSpaceCount | |
| 26 | GcNotConnectedCount | |
| 27 | GcNotConnectedDeadCount | |
| 28 | IntSendCount (obsolete) | |
| 29 | ClassCacheCount (obsolete) | |
| 30 | MethodCacheCount (obsolete) | |

(Continued)

**Table 10.3 Cache Statistics (Continued)**

| Array Element[a] | Name | Cache Slots for Which Available |
|---|---|---|
| 31 | ExportedSetSize | Gems only |
| 32 | NoRollbackSetSize | |
| 33 | NotConnectedObjsSetSize | |
| 34 | TotalCommits | Stone only |
| 35 | CommitRecordCount | |
| 36 | AsyncWritesInProgress | |
| 37 | AsyncWritesCount | |
| 38 | LogRecordsWritten | |
| 39 | LogRecordsIoCount | |
| 40 | EpochGcCount | |
| 41 | DeadObjsCount | |
| 42 | ReclaimCount | |
| 43 | ReclaimedPagesCount | |
| 44 | CheckpointCount | |
| 45 | AioDirtyCount | Page servers only |
| 46 | AioCkptCount | |
| 47 | LocalDirtyPageCount | Shared page cache monitor only |
| 48 | GlobalDirtyPageCount | |
| 49 | PagesNeedReclaimSize | Stone only |
| 50 | PossibleDeadSize | |
| 51 | DeadNotReclaimedSize | |
| 52 | InTransaction | Gems only |
| 53 | TimeWaitingForCommit | |
| 54 | TimeProcessingCommit | |
| 55 | TimeStoneCommit | |
| 56 | CommitQueueSize | Stone only |
| 57 | LockReqQueueSize | |
| 58 | NotifyQueueSize | |
| 59 | LoginWaitQueueSize | |
| 60 | RunQueueSize | |

(Continued)

**Table 10.3 Cache Statistics (Continued)**

| Array Element[a] | Name | Cache Slots for Which Available |
|---|---|---|
| 61 | PageKindsWrittenByGems (see Table 10.4) | Shared page cache monitor only |
| 62 | PageKindsWrittenByStone (see Table 10.4) | |
| 63 | MilliSecPerIoSample | Stone only |
| 64 | AllSymbolsQueueSize | |
| 65 | PageWaitQueueSize | |
| 66 | LogWaitQueueSize | |
| 67 | TempPagesDisposed | |
| 68 | PersistentPagesDisposed | |
| 69 | PageDisposesDeferred | |
| 70 | SigAbortCount | Gems only |
| 71 | SigLostOtCount | |
| 72 | MessagesToStone | |
| 73 | ProgressCount | |
| 74 | NewSymbolsCount | |
| 75 | BytesCommittedCount | |
| 76 | FreePages | Stone and Gems only |
| 77 | ClientPageReads | Page servers only |
| 78 | ClientPageWrites | |
| 79 | VcCacheNumScavenges | Gems only |
| 80 | VcCacheSizeBytes | |
| 81 | CodeCacheSizeBytes | |
| 82 | CodeCacheNumEntries | |
| 83 | CodeCacheNumStaleEntries | |
| 84 | CodeCacheNumScavenges | |
| 85 | FramesFromFreeList | All |
| 86 | FramesFromFindFree | |

(Continued)

**Table 10.3 Cache Statistics (Continued)**

| Array Element[a] | Name | Cache Slots for Which Available |
|---|---|---|
| 87 | GcPromoteDeadCount | Stone only |
| 88 | GcSweepCount | |
| 89 | GcPossibleDeadSize | |
| 90 | GcPossibleDeadWSUnionSize | |
| 91 | GcPagesNeedReclaiming | |
| 92 | GcDeferPromoteDeadThreshold | |
| 93 | GcDeferEpochThreshold | |
| 94 | GcReclaimMaxPages | |
| 95 | GcReclaimNewDataPagesCount | |
| 96 | VoteNotDead | Gems only |

[a] Offsets may change in future releases. For current offsets, see the method `System | cacheStatisticsDescription`.

The following descriptions are in alphabetical order. The number in parentheses is the array element in Table 10.3.

## AbortCount

AbortCount (17) is the number of aborts executed by a Gem process (or by an application linked to a Gem) since the Gem was most recently started.

## AioCkptCount

AioCkptCount (46) is the number of dirty pages written from the shared page cache to disk to satisfy a checkpoint.

## AioDirtyCount

AioDirtyCount (45) is the number of dirty pages written from the shared page cache to disk by Stone's AIO page server during normal operation.

## AllSymbolsQueueSize

AllSymbolsQueueSize (64) is 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.

## AsyncWritesCount

AsyncWritesCount (37) is the number of asynchronous writes performed since the process started (only applies to the Stone repository monitor process).

## AsyncWritesInProgress

AsyncWritesInProgress (36) is the number of outstanding asynchronous writes queued (only applies to the Stone repository monitor process).

## AttachDelta

AttachDelta (5) is 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.

The value for AttachDelta is periodically recalculated by the cache monitor process. Similar (but not necessarily constant) values for active processes performing similar tasks are an indicator that the cache is being shared and that the page cache monitor process is working properly. A process that is idle may have a negative value, or a small positive value quite different from other sessions.

## AttachedCount

AttachedCount (6) is 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.

## BytesCommittedCount

BytesCommittedCount (75) is the total number of bytes that have been committed by this session.

### CheckpointCount

CheckpointCount (44) is 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.

### ClassCacheCount

ClassCacheCount (29) is obsolete and always returns zero.

### ClientPageReads

ClientPageReads (77) 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

ClientPageWrites (78) 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.

### CodeCacheEntries

CodeCacheEntries (77) is the number of entries in the code cache.

### CodeCacheScavengesCount

CodeCacheScavengesCount (79) is the number of garbage collections of the code cache.

### CodeCacheStaleEntries

CodeCacheStaleEntries (78) is the number of stale methods in the code cache.

### CodeCacheSizeBytes

CodeCacheSizeBytes (76) is 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.

### CommitCount

CommitCount (15) is the number of commits executed by a Gem process (or application linked to a Gem) since the Gem was most recently started.

### CommitQueueSize

CommitQueueSize (56) shows the number of Gem session processes waiting for the commit token.

### CommitRecordCount

CommitRecordCount (35) is 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.

### DeadNotReclaimedSize

DeadNotReclaimedSize (49) is 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.

### DeadObjCount

DeadObjCount (24) is the number of dead objects that have been garbage collected by the in-memory garbage collection of temporary objects since the process started.

### DeadObjsCount

DeadObjsCount (41) is 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.

### EpochGcCount

EpochGcCount (40) is 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.

### ExportedSetSize

ExportedSetSize (31) is 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.

### FailedCommitCount

FailedCommitCount (16) is the number of attempts to commit that failed due to concurrency conflicts.

### FramesFromFreeList

FramesFromFreeList (85) is the number of times the process acquired a page frame in the shared page cache from the list of free frames.

### FramesFromFindFree

FramesFromFindFree (86) is 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.

### FreeFrameCount

FreeFrameCount (9) is 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.

### FreePages

FreePages (76) is the size of the free page pool for the repository. Free space in the repository is calculated at 8 KBytes for each page in the free pool.

### GcDeferEpochThreshold

GcDeferEpochThreshold (93) is the value of the GcUser parameter `#deferEpochReclaimThreshold` that the GcGem is currently using.

### GcDeferPromoteDeadThreshold

GcDeferPromoteDeadThreshold (92) is the value of the GcUser parameter `#deferPromoteDeadReclaimThreshold` that the GcGem is currently using.

### GcNotConnectedCount

GcNotConnectedCount (26) is the number of times the notConnected objects were garbage collected since the process started.

### GcNotConnectedDeadCount

GcNotConnectedDeadCount (27) is the number of dead objects found during the garbage collection of the notConnected objects.

### GcPagesNeedReclaiming

GcPagesNeedReclaiming (91) is 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 GcDeferPromoteDeadThreshold (92), then promotions are deferred. If this value is greater than GcDeferEpochThreshold (93), then epochs are deferred.

### GcPossibleDeadSize

GcPossibleDeadSize (89) is 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

GcPossibleDeadWSUnionSize (90) is 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 (73) during a GcGem sweep may reach this value as its maximum, this value can be used to estimate when the sweep will complete.

### GcPromoteDeadCount

GcPromoteDeadCount (87) is the total number of objects that the GcGem has promoted to dead.

### GcReclaimMaxPages

GcReclaimMaxPages (94) is the value of the GcUser parameter `#reclaimMaxPages` that the GcGem is currently using.

### GcReclaimNewDataPagesCount

GcReclaimNewDataPagesCount (95) is the number of new data pages that the GcGem had to allocate during reclaims since the Stone process was started.

### GcSweepCount

GcSweepCount (88) is the total number of sweeps of the possible dead write set union that have been done by the GcGem since it started.

### GlobalDirtyPageCount

GlobalDirtyPageCount (48) is 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).

### InTransaction

InTransaction (52) indicates whether the Gem process is in a transaction.

### IntSendCount

IntSendCount (28) is obsolete and always returns zero.

### LocalDirtyPageCount

LocalDirtyPageCount (47) is 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

LocalPageCacheHits (10) is 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

LocalPageCacheMisses (11) is 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

LocalPageCacheWrites (12) is 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

LockedPage (4) is obsolete and always returns zero.

### LockReqQueueSize

LockReqQueueSize (57) shows the number of Gem session processes waiting for a commit to complete so that their lock request can be serviced.

### LoginWaitQueueSize

LoginWaitQueueSize (59) shows the number of sessions waiting for login completion.

### LogRecordsIoCount

LogRecordsIoCount (39) is 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

LogRecordsWritten (38) is 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

LogWaitQueueSize (66) is 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.

## MakeRoomInOldSpaceCount

MakeRoomInOldSpaceCount (25) is number of times the oldest temporary object generation filled up. Large values are okay 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 (33) to see if both are growing. Another useful comparison is that MakeRoomInOldSpaceCount should be less than one-third the size of ScavengeCount (22); larger ratios indicate that the cache is too small.

## MessagesToStone

MessagesToStone (72) is the number of messages sent by the Gem session process to the Stone repository monitor using shared memory as the channel.

## MethodCacheCount

MethodCacheCount (30) is obsolete and always returns zero.

### MilliSecPerIoSample

MilliSecPerIoSample (63) is used as a parameter to implement the configurable I/O limit for GemStone processes. Because a process's I/O rate currently is sampled every 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 primitive. For information about setting this value, see the discussion on page 2-15. If you want to calculate the current I/O limit, it is given by 1000/(MilliSecPerIoSample * 5).

### NewObjsCommitted

NewObjsCommitted (19) is the number of new objects committed by the most recent transaction committed by this process.

### NewSymbolsCount

NewSymbolsCount (74) is the number of new symbols created by this session.

### NoRollbackSetSize

NoRollbackSetSize (32) is 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

NotConnectedObjsSetSize (33) is 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

NotifyQueueSize (58) shows the number of Gem session processes using notifiers.

### ObjsCommitted

ObjsCommitted (18) is the number of objects committed by the most recent transaction committed by this process.

### PageDisposesDeferred

PageDisposesDeferred (69) is 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.

## PageKindsWrittenByGems
## PageKindsWrittenByStone

PageKindsWrittenByGems (61) and PageKindsWrittenByStone (62) each consist of an array of counts for 19 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 10.4 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 10.4 Page Kinds in Shared Page Cache**

| Index | Page Kind |
|---|---|
| 1 | Invalid (empty) page |
| 2 | Root page |
| 3 | (for internal use) |
| 4 | Commit record page |
| 5 | Data page |
| 6 | Object Table internal page |
| 7 | Object Table leaf page |
| 8 | Bitmap internal page |
| 9 | Bitmap leaf page |
| 10 to 12 | (for internal use) |
| 13 | Bitlist page (a form of bit array) |
| 14 to 19 | (for internal use) |

## PagesNeedReclaimSize

PagesNeedReclaimSize (49) is 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.

## PageReads

PageReads (13) is 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.

### PageWaitQueueSize

PageWaitQueueSize (65) is 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

PageWrites (14) is 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.

### PersistentPagesDisposed

PersistentPagesDisposed (68) is the number of persistent pages (pages already checkpointed) that have been disposed of while in the Stone's private cache.

### PossibleDeadSize

PossibleDeadSize (50) is 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.

### ProcessId

ProcessId (2) is the operating system processId for the process associated with this shared page cache process slot.

### ProcessName

ProcessName (1) identifies the process kind (Gem, Stone, page server, or shared page cache monitor).

### ProgressCount

ProgressCount (73) 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).

### ReclaimCount

ReclaimCount (42) is the number of reclaims performed by a GcGem process since the Stone repository monitor was last started.

### ReclaimedPagesCount

ReclaimedPagesCount (43) is 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.

### RunQueueSize

RunQueueSize (60) shows the number of Gem session processes waiting for service from the Stone repository monitor.

### ScavengeCount

ScavengeCount (22) is the number of times that the in-memory temporary object garbage collector was executed since the process started.

### SessionId

SessionId (3) is the GemStone sessionId associated with this client.

### SharedAttached

SharedAttached (7) is 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

SigAbortCount (70) is 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.

### SigLostOtCount

SigLostOtCount (71) is the number of lost object table signals that have been sent by the Stone to this session.

### TempPagesDisposed

TempPagesDisposed (67) is the number of temporary pages (pages allocated since the last checkpoint) that have been disposed.

### TimeInScavenges

TimeInScavenges (23) is CPU time in milliseconds spent in the in-memory temporary garbage collector.

### TimeProcessingCommit

TimeProcessingCommit (54) shows 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

TimeStoneCommit (55) shows 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

TimeWaitingForCommit (53) shows 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.

### TotalAttached

TotalAttached (8) is 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

TotalCommits (34) is the total number of commits (excluding read-only commits) performed by all processes since the Stone repository monitor was last started.

### TotalNewObjsCommitted

TotalNewObjsCommitted (21) is obsolete and always returns zero.

### TotalObjsCommitted

TotalObjsCommitted (20) is obsolete and always returns zero.

### VcCacheScavengesCount

VcCacheScavengesCount (74) is the number of garbage collections of the VC space. VC space is a memory region private to the virtual machine.

### VcCacheSizeBytes

VcCacheSizeBytes (75) is the total size in bytes of the VC space. This space is the private heap memory of the Gem.

### VoteNotDead

VoteNotDead (96) is the number of objects that the Gem process removed from the possibleDead set the last time that it voted on the possibleDead.

GemStone

# Part III: Concepts for the System Administrator

Chapter 11

# GemStone Garbage Collection

This chapter describes the garbage collection techniques implemented in GemStone 5.0.

## 11.1 Introduction

*Garbage collection* is the automatic reclamation of computer storage [Knu69, Wils92]. An object is considered *live* in GemStone if it is reachable by traversing some path from a reserved object. Because one of these reserved objects is AllUsers, each user's symbol list is included in the path. The remaining objects, which cannot be reached, are candidates for reclamation.

### Motivation

Smalltalk execution produces a number of objects that are no longer needed, both temporary objects and previously committed objects that are no longer referenced. Windowed interfaces create numerous temporary objects in the course of screen management. To make the best use of system resources, it is desirable to reclaim the storage these objects occupy as soon as possible.

## Terminology

The following terms are used in this chapter:

**dead object —** an object for which no reference exists in the repository and which is now visible only to the system.

**dispose (CR) —** the process of discarding the internal record of a committed transaction, or commit record (CR). GemStone must maintain a given commit record as long as that record is the basis of some active session's transaction view of the repository. For sessions outside of a transaction, the record must be maintained at least as long as a configurable backlog. When these needs have passed, GemStone can dispose of the commit record and act on any garbage collection information it contains.

**generational scavenging —** a kind of scavenging that concentrates on the most recently created objects on the principle that only a few of these objects will be alive during the first garbage collection. The survivors are moved to a series of memory spaces containing progressively older generations.

**marked objects —** objects that GemStone has determined to be alive during the first phase of garbage collection. The remaining, *unmarked* objects are possible candidates for reclaiming. See *possible dead object* and *voting*.

**possible dead object —** a committed object that has since been dereferenced by a committed transaction but for which other, concurrent transactions might have created a reference. See *voting*.

**reclaim —** the process of analyzing pages containing dead or shadowed objects and preparing those pages and the old object identifiers for reuse. Live objects are moved to another page.

**shadowed object —** the previous value of a committed object. A committed object becomes a shadowed object when it is modified during a transaction. The shadowed object must be maintained as long as it is visible to transactions that were concurrent with the transaction that committed the change. Unlike a dead object, a shadowed object is still referenced in the repository because the old and new values share a single object identifier.

**voting —** the process by which each active session indicates whether it has created a reference to a committed object that has been marked as possibly dead. Because GemStone supports concurrent sessions that can see the same objects, one session can create a reference to an object that another session just

dereferenced. Each session votes the next time it commits or aborts a transaction.

## Scope

This chapter addresses the garbage collection of *dead* objects and the reclamation of *shadowed* objects. Dead objects are objects that are dereferenced and not reachable from AllUsers or other root objects. GemStone transactions also leave shadowed objects, which are previously committed objects that have been modified. Both the shadowed object and its successor are referenced, and they bear the same identifier. The shadowed object must be maintained for a time to support multiple, independent transaction views.

The following examples illustrate the difference between dead and shadowed objects. Notice that collection of a dead object reclaims both its space and its identifier (oop), while collection of a shadowed object reclaims only its space.

The first example (Figure 11.1) 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, 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 11.1 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
```

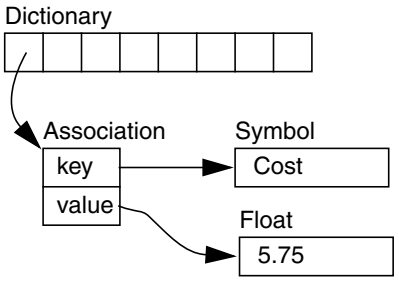



Figure 11.2 shows a second Topaz session that logs in at this point. Notice the Topaz prompt identifies the session by displaying a digit. Because Session 1 committed the SymbolAssociation to the repository, Session 2 can see the SymbolAssociation.

**Figure 11.2 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>
```

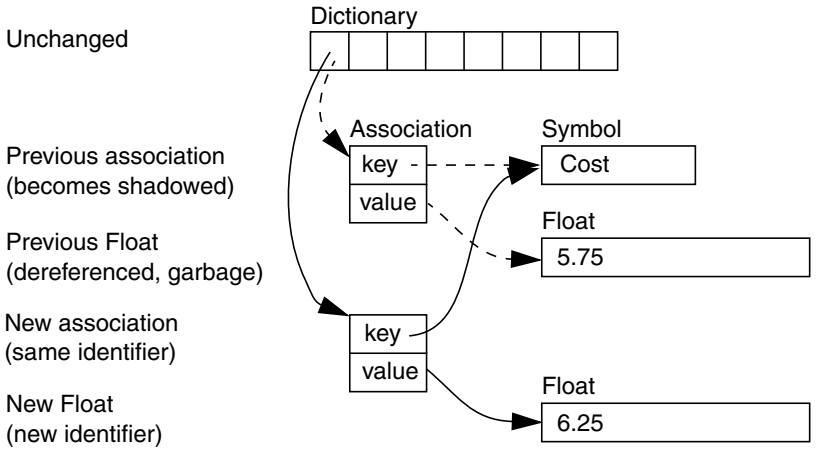
If Session 1 changes the value instance variable, a new SymbolAssociation is created (Figure 11.3). Notice in the oops display that the new SymbolAssociation object has the same identifier (126321) as the previous Association.

- The previous SymbolAssociation is now *shadowed*. Because the shadowed SymbolAssociation was part of the committed repository and may still be to visible other transactions, it cannot be overwritten. Instead, the new SymbolAssociation is written to another page that is 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*.

**Figure 11.3 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 11.4). When Session 2 (and any other concurrent session) either commits or aborts the transaction that was underway at the time Session 1 committed the change, it sees the new SymbolAssociation and value. When all sessions have committed or aborted their concurrent transaction, the previous objects can be collected as garbage.

**Figure 11.4 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
```

## Five Collection Mechanisms

GemStone 5.0 provides five mechanisms for garbage collection to eliminate dead objects as soon as possible. Three stages automatically perform garbage collection in memory or shortly after transactions are written to the disk. The final two stages can be invoked manually to sweep the repository.

**Collection in Local Object Memory —** The first collection mechanism, which you might think of as preventive, reclaims objects in local object memory (LOM) while they exist within view of only the session that created them. Each Gem session process performs this task automatically. The goal is to keep temporary garbage objects from reaching permanent object memory (POM) through either a committed transaction or an overflow of temporary object space.

**Collection of the NotConnectedSet in Permanent Object Memory —** The second mechanism examines objects that the Gem session process has moved to POM but which are not referenced by permanent objects. Each Gem performs a mark-sweep on this *notConnectedSet* when circumstances warrant and releases the dead objects for the next garbage collection in POM.

**Epoch Collection in Permanent Object Memory —** The third mechanism examines objects shortly after they are committed to POM, which can be a page on a disk or a page frame in the shared page cache. All transactions written since a specific, recent time (the beginning of this *epoch*) are screened for objects that were created and then dereferenced during that period. The GcUser session performs this task periodically. However, objects that are created in one epoch but dereferenced in another will not be reclaimed by the epoch collection mechanism.

**Targeted Collection in Permanent Object Memory —** The mechanism can be invoked by a GemStone administrator to reclaim specific objects that the application believes to be dead. The application adds these objects to the global queue GcCandidates. The administrator initiates their collection by invoking `Repository | markGcCandidates` in a GemStone Smalltalk session. This method sweeps the repository and reclaims those objects in GcCandidates that have no other reference.

**Complete Collection in Permanent Object Memory —** The final and broadest mechanism should be invoked occasionally by a GemStone administrator to reclaim any dead objects that slipped through the other collection mechanisms. The administrator initiates this collection by invoking `Repository | markForCollection`. Because this method marks dead objects anywhere in the entire repository, it reclaims objects that eluded detection within an epoch and were not added to GcCandidates.

## Identifying Live Objects

Most GemStone garbage collection mechanisms involve at some point a *transitive closure* scan as part of a mark-sweep to identify live objects. This scan begins with a *root set* of objects and traverses each reference it encounters until no more references are found. Each object in the root set and each object encountered while traversing references is marked as being live and is added to a *marked* set, which becomes the output of the transitive closure. What constitutes the root set depends on the particular garbage collection mechanism.

# 11.2 Reclaiming Local Object Memory

Local object memory (LOM) exists in each session's private memory space. GemStone performs object garbage collection in LOM to maximize its use and to minimize the number of dead objects that find their way to the disk.

When a session creates a new object, GemStone allocates space for it in a work space in that Gem's LOM. (Large objects, those occupying more than a single 8 KByte page, are written directly in Permanent Object Memory, or POM.) Objects that survive a generational scavenge in the work space eventually migrate to the Gem's *temporary object space*. The temporary object space is configurable, and you may want to tune its size to the needs of your application.

If the temporary object space becomes full, 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 the temporary object space are moved directly to POM, where they become part of the Gem's notConnectedSet. If the pages containing these objects are committed, future garbage collection must be handled by POM techniques, which are more costly.

## Parameters

The GEM_TEMPOBJ_CACHE_SIZE configuration option determines the temporary object space, which by default is 585 KBytes. Sessions that create a large number of temporary objects may need to increase its size to avoid cluttering POM with objects that have a short life expectancy.

## Statistics

The method `System | cacheStatistics:` for a Gem's cache slot reports several statistics that are useful for monitoring LOM garbage collection:

`NumberOfScavenges` — the number of times generational scavenge was invoked in this session's work space

`TimeInScavenges` — the total CPU time in seconds spent in generational scavenge during this session

`NumberOfDeadObjects` — the number of oops reclaimed by generational scavenge in this session

`NumberOfMakeRoomInOldSpace` — the number of mark-sweeps of temporary object space to make room for younger objects

## 11.3 Reclaiming the NotConnectedSet

After each generational scavenge of LOM, the Gem session process examines the total number objects in its notConnectedSet. These objects are ones that the Gem moved to POM even though they were not referenced by committed objects. If the size of this set meets the `#NotConnectedThreshold` and `#NotConnectedDelta` parameters (see page 2-13), the Gem performs a mark-sweep on the set. Any objects referenced from LOM are retained in the notConnectedSet and thereby protected from garbage collection. The remaining objects are garbage collected by the Gem, or (if the page has been committed) are made available for the next epoch garbage collection in POM.

Removing objects from a Gem's notConnectedSet provides two benefits:

- The objects are made available to epoch garbage collection even though the session is still logged in.
- The overhead for voting (page 11-12) is reduced because these objects have not become visible to other sessions.

For tuning information, see "To Tune Garbage Collection of the NotConnectedSet" on page 2-12.

## 11.4 Reclaiming Permanent Object Memory

Garbage collection in Permanent Object Memory (POM) differs substantially from the generational approach used in LOM:

- POM uses 8 KByte pages as the basic read-write-reclaim unit. Unlike generational scavenging in LOM, objects in POM are not garbage collected individually. Instead, the presence of a shadowed or dead object triggers reclamation of the page on which it resides.
- POM garbage collection is closely tied to repository *commit records* and the need to support multiple, independent transaction views of the repository. Each session's transaction view is based on exactly one commit record at a time, but any number of sessions can be based on the same commit record. *Each commit record and the objects to which it refers must be retained in the repository as long as that commit record defines the transaction view of some session.*

This section first describes the reclamation cycles through which data pages and objects are made available for reuse. Then it describes the three ways by which the object reclamation cycle can be initiated: two methods in class Repository,

`markForCollection` and `markGcCandidates`, and epoch garbage collection, which runs automatically.

## Reclaiming Pages

Figure 11.5 shows the reclamation cycle for pages, beginning with a committed transaction at the top. The commit record includes a list of pages that are reclaimable because they contain at least one shadowed object—the current object is on a new page. (Commit records may also list pages containing dead objects, but that process requires additional steps that are discussed later.)

The page reclamation cycle takes place in three time intervals ($I_n$) of variable length, which are shown in the time line at the bottom of the illustration. The interval labels are circled in Figure 11.5.

$I_1$ The commit record (CR) stays alive for a variable period until neither it nor any older commit record is the basis for some session's view of the repository. For sessions in a transaction, that means waiting until they either commit or abort. For sessions outside a transaction, the period depends on the STN_SIGNAL_ABORT_CR_BACKLOG configuration option, which sets the number of commits that must intervene before the Stone can force a sleeping session to update its view of the repository.

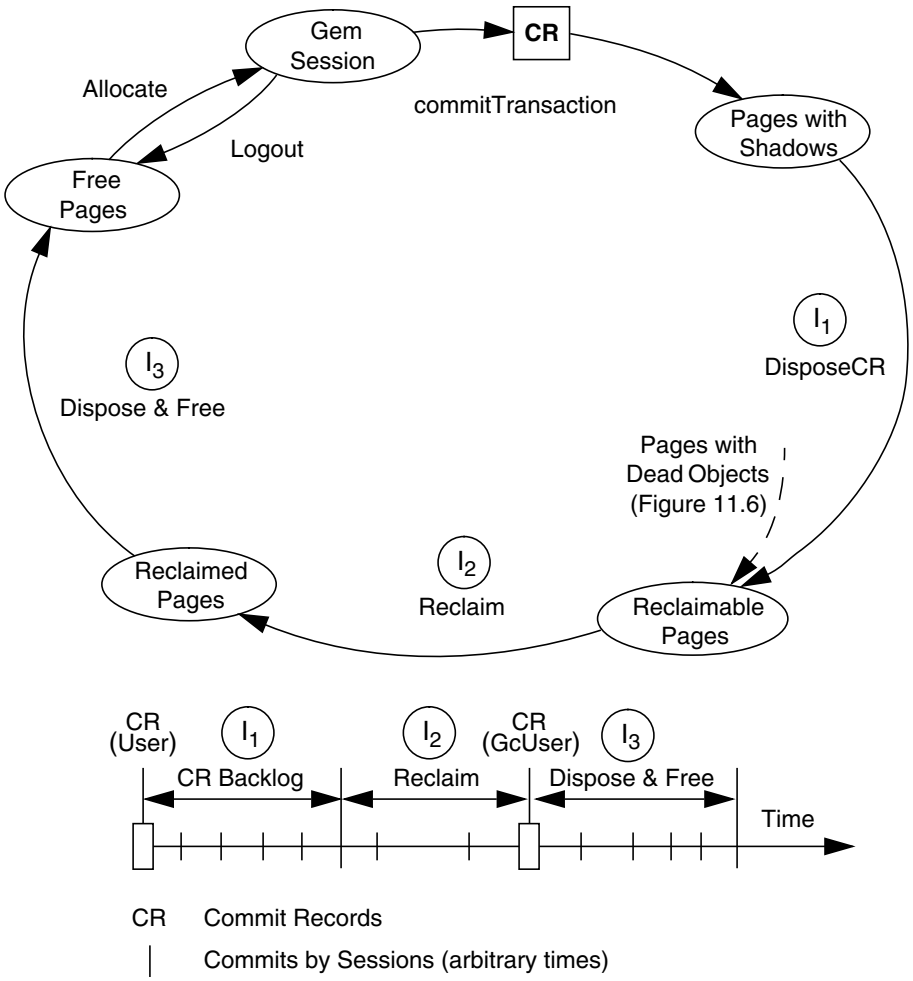
Once the need to maintain the commit record expires, the Stone can add the pages with shadowed objects to a global list of reclaimable pages.

$I_2$ GcUser runs the reclamation process whenever there are a sufficient number of reclaimable pages in the global list. For each reclaimable page, Reclaim classifies valid objects into one of three categories and acts on the objects accordingly:

- Live objects have their current representation on this page. These objects are copied forward to a new page, thereby compacting live objects in page space. (To limit extent fragmentation, the Gem also seeds the list of reclaimable pages with recently created pages that have 10% or more free space. Small transactions often occupy less than the minimum POM allocation of one page.)
- Objects known to be dead (that process is described later) have their identifiers added to a list of reclaimed oops.
- Shadowed objects already have their current representation on another page; because the representations share the same oop, there is no oop to reclaim. No action is necessary.

$I_3$ The page now contains only garbage and can be freed as a unit. The time interval before the page shows up as free space is variable because it depends on disposal of GcUser's commit record and on the timing of repository checkpoints.

**Figure 11.5 Reclamation Cycle for Pages**



## Reclaiming Objects

The reclamation cycle for dead objects adds two steps to those necessary for reclaiming pages that contain only shadowed objects. Figure 11.6 shows the object cycle, again beginning with a committed transaction at the top. The objects in this transaction become visible to any other session that subsequently logs in, commits, or aborts.

The object reclamation cycle takes place in five intervals that correspond to the circled labels in Figure 11.6:

$I_1$ Object reclamation begins with the marking of possible dead objects in the repository. The GcUser session does this periodically as part of epoch garbage collection, and users who have the GarbageCollection privilege can do it by invoking `Repository | markForCollection` or `markGcCandidates`. (The details are presented later in this chapter.)

When the number of pages waiting to be reclaimed exceeds the threshold set by `#deferEpochReclaimThreshold` , epoch garbage collection is suspended. For more information, see page 7-23.

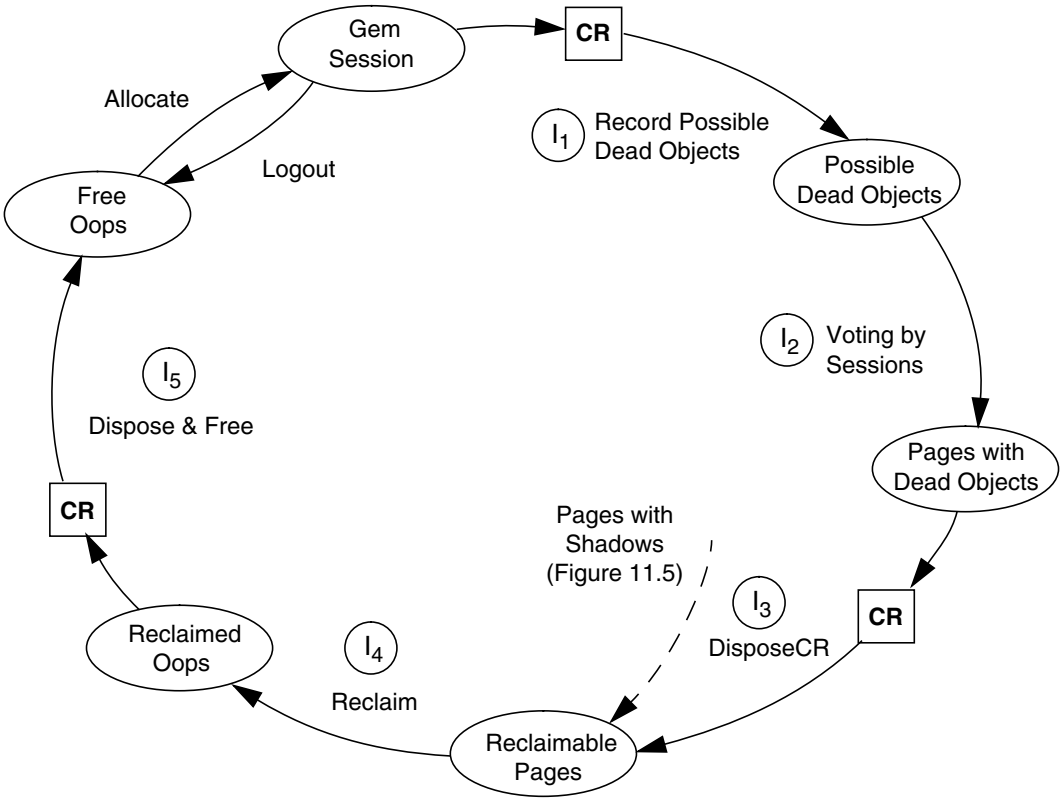
$I_2$ At this point, the possible dead objects are still visible to other GemStone sessions. A session in a transaction started prior to the time objects are recorded possibly dead continues to “see” these objects until it either commits or aborts. Before garbage collection can proceed, each active GemStone session must “vote” on the possible dead objects after examining its cache for any reference to them. This voting occurs at the transaction boundary marked by a commit or an abort.

Once all active sessions have voted on an object, the GcUser declares it dead by flagging its identifier and acquiring an exclusive lock on it so that no session can create a reference to it. Then the GcUser adds the object’s data page to the list of pages that can be reclaimed and produces a commit record (CR) containing this information. The dead object is no longer visible to user sessions based on that commit record or a more recent one.

The maximum number of possibly dead objects that GcUser can declare dead during a single run is set by `#promoteDeadLimit` in GcUser’s UserGlobals. That parameter provides a way of partitioning the work load for the reclaim task. The default limit is 1000 objects.

When the number of pages waiting to be reclaimed exceeds the threshold set by `#deferPromoteDeadReclaimThreshold` , action declaring additional objects dead is suspended. For more information, see page 7-23.

**Figure 11.6 Reclamation Cycle for Oops**



$I_3$ From this point on, the reclamation cycle for dead objects is much the same as for pages, beginning with a variable delay until the commit record is no longer needed.

$I_4$ When GcUser runs the reclamation process, objects flagged as dead have their identifiers reclaimed as described on page 11-10.

$I_5$ After a variable delay like that for pages, the identifiers are added to the free list.

The STN_SIGNAL_ABORT_CR_BACKLOG configuration option determines the number of transaction commits that must take place before the Stone can dispose of a commit record being used by a session outside of a transaction and act upon the scavenging information it contains. This option is used to balance the number of commit records (transactions) maintained in memory against the system swapping activity brought about by awakening inactive sessions to update their view of the repository.

For administrative procedures and tuning information, see "To Reclaim Pages" on page 7-20.

## markForCollection

The method `Repository | markForCollection` can be invoked from a transaction by a user who has the GarbageCollection privilege. This method sweeps the entire repository and marks as live all objects that can be reached through a transitive closure on the symbol lists in AllUsers. The remaining objects become the list of possible dead objects. In terms of the transitive closure method described on page 11-7, the following definitions apply:

| | |
|---|---|
| RootSet | reserved oops plus the symbol lists in AllUsers |
| Universe | all committed objects at the time this transaction began (that is, objects in subsequent transactions are excluded) |
| PossibleDead | Universe minus Marked |

Recall that `markForCollection` only provides a set of possible dead objects for voting and eventual reclaiming as described under "Reclaiming Objects" on page 11-12. It does not reclaim the space or identifiers itself.

For administrative procedures and performance information, see "To Run markForCollection" on page 7-17.

## markGcCandidates

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.

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.

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 like that described on page 11-7 is performed only on the live objects found in GcCandidates. As a result, this method can mark specific objects as possible dead more efficiently than `markForCollection`, which performs a transitive closure on the entire repository.

Recall that all marking activities only provide a set of possible dead objects for voting and eventual reclaiming, as described under “Reclaiming Objects” on page 11-12. They themselves do not reclaim the space or identifiers.

For procedures, see “To Run markGcCandidates” on page 7-16.

## An Example of Marking and Reclaiming Objects

The following example illustrates a `markForCollection` and the eventual reclamation of disk space occupied by the objects. The emphasis is on the interaction of transactions by three users:

- DataCurator, whose only task is to invoke `markForCollection`;
- SomeUser, who represents ongoing application activity by committing occasional but timely changes to the repository; and
- GcUser, who performs garbage collection tasks related to the promotion of dead objects and the reclaiming of pages.

At the beginning of this example, a large Array has been dereferenced in the repository and is ready for collection.

One GemStone configuration option is changed from its default setting: STN_SIGNAL_ABORT_CR_BACKLOG is set to 0 (instead of 20) so that unneeded commit records can be disposed quickly for demonstration purposes. Ordinarily,

a larger backlog would be maintained to support sessions outside of a transaction without excessive swapping. *The setting used in this example would be both unnecessary and impractical in a production environment in which there is a stream of commits by other sessions.*

To analyze the collection process, this example uses selected elements of an array returned by `System | cacheStatistics:`

#43 **ReclaimedPagesCount**, the number of pages reclaimed by GcUser since the Stone last started.

#76 **FreePages**, the size of the free page pool for the repository. Free space in the repository is calculated at 8 KBytes for each page in the free pool.

#87 **GcPromoteDeadCount**, the total number of objects that the GcGem has promoted to dead.

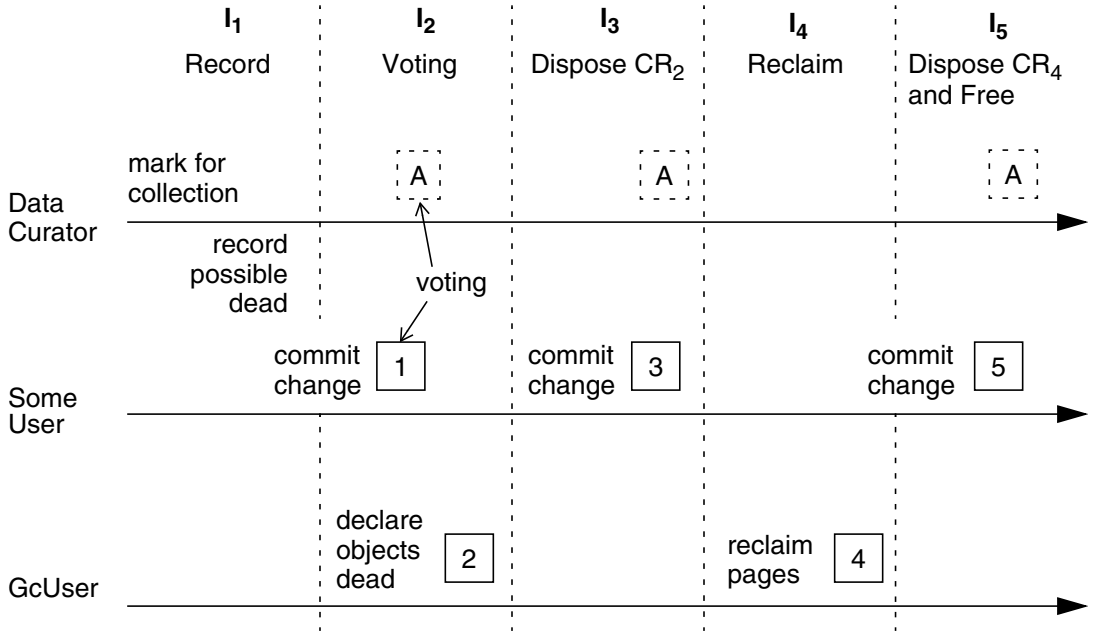
#89 **GcPossibleDeadSize**, the number of possible dead objects remaining. Initially, it is the size of the possible dead set received by the GcGem from the Stone. Each time a group of objects is promoted, this value is decremented by the size of that group.

#91 **GcPagesNeedReclaiming**, the number of pages that currently need reclaiming. A large backlog may cause promotion of additional objects or epoch garbage collection to be deferred.

For a general description of the cache statistics, see “To Monitor Page Reads and Writes by a Session” on page 10-7.

Figure 11.7 shows the interaction between the three sessions in this example. Their activity is divided into five intervals, which are labeled $I_n$. A commentary and Topaz example follow for each interval.

**Figure 11.7 Timing of the markForCollection Example**




A — An aborted transaction, which updates that session's transaction view of the repository to its current committed state.

*n* — Commit record *n*, which becomes that session's view of repository.

## Interval $I_1$ — Record Possible Dead

DataCurator invokes `Repository | markForCollection` . See the Topaz commands and the selected output in Example 11.1. When the method completes sweeping the repository, it records the dereferenced objects. Cache statistics for the Stone's process slot show there are 137 free pages in the repository at this point, or about 1.1 MBytes.

The actual garbage collection cannot begin until all active user sessions either commit or abort the transactions that were concurrent with the `markForCollection` .

**Example 11.1 Interval 1, Recording Possible Dead Objects**

```
topaz 1> printit
                                        "Identify the GemStone sessions"
System currentSessionNames
%
session number: 1  UserId: GcUser
session number: 2  UserId: DataCurator      [Topaz session 1]
session number: 3  UserId: SomeUser         [Topaz session 2]
topaz 1> printit
SystemRepository markForCollection
%
Successful completion of markForCollection.
    33937 live objects found.
    2503 dead objects, occupying 20067608 bytes, will be reclaimed .
topaz 1> printit
System cacheStatistics:1
%
an Array
  #76 137    [FreePages]
```

## Interval $I_2$— Voting

During the second interval, each active session either commits or aborts its transaction. This transaction boundary is the point at which each session "votes" by indicating whether its workspace contains a reference to the possible dead objects. In our example, SomeUser commits a change to the repository ($CR_1$ in the example), and then DataCurator aborts.

Although these transitions occur one after the other in our example, in a typical scenario they might be widely separated and interspersed with other actions. The method `System | sessionsReferencingOldestCr` can be used to identify sessions holding on to the oldest commit record.

Because the user sessions in our example had not created references to the possible dead objects, GcUser can now obtain exclusive locks and declare them dead. GcUser then promotes the dead objects and their pages in a new commit record, $CR_2$.

Notice at the end of this interval that GcUser has promoted 2000 objects from possible dead to dead (#87) and lists 2021 pages as needing reclamation (#91). Another 440 objects remain as possible dead (#89) for processing later. This

snapshot illustrates the effect of two configurable limits. The GcUser parameter #promoteDeadLimit by default limits promotion to 1000 objects at a time. After the first 2000 objects have been promoted (two groups), the total of 2021 pages awaiting reclamation (#91) has exceeded another parameter, #deferPromoteDeadReclaimThreshold . (The value of this parameter currently being used is available as #92.) Further promotion is suspended until the reclaim backlog falls below the threshold.

**Example 11.2 Interval 2, Voting on Possible Dead Objects**

```
topaz 1> # Commit or abort both sessions so they vote at boundary.
topaz 1>  set session 2
topaz 2>  printit
       "SomeUser commits, in the process voting on possible dead
        also creating a new commit record."
UserGlobals at: #UserCommits put: 1.
System commitTransaction.
%
true
topaz 2>  set session 1
topaz 1> # Advance to new CR, voting in the process
topaz 1>  abort
topaz 1>  printit
       "Let GcUser do its thing"
System sleep: 10
%
topaz 1>  printit
System cacheStatistics:1
%
an Array
  #87 2000   [GcPromoteDeadCount]
  #89 440    [GcPossibleDeadSize]
  #91 2021   [GcPagesNeedReclaiming]
  #92 1500   [GcDeferPromoteDeadThreshold]
```

## Interval $I_3$ — Dispose of CR

Before GcUser can process the list of dead objects and their pages, $CR_2$ must no longer be needed. For that to happen, some session must first commit a change to the repository — which SomeUser does as $CR_3$. That commit moves SomeUser's transaction view passed $CR_2$, and the commit record's existence permits DataCurator to do the same by aborting.

At the end of this interval, $CR_2$ is no longer needed.

In our example, only a single transaction is necessary because STN_SIGNAL_CR_BACKLOG was set to 0. Ordinarily, ongoing application activity would supply the commit records necessary for sessions to progress through the backlog specified by the configuration option.

**Example 11.3 Interval 3, Disposing of the Commit Record**

```
topaz 1>  set session 2
topaz 2>  printit
       "Need another CR so CR2 can be disposed.
        Have SomeUser commit another change to repository."
UserCommits := 2.
System commitTransaction
%
true
topaz 2>  set session 1
topaz 1> # Advance DataCurator's transaction view to the new CR too.
topaz 1>  abort
```

## Interval $I_4$ — Reclaim Pages

GcUser can now dispose of $CR_2$ and reclaim the pages. Because the number of pages was greater than `#reclaimMaxPages` , GcUser performs the reclamation in two passes, reclaiming a total of 2021 pages (#43). The reclaimed pages are recorded in $CR_4$, which here represents two commit records, one for each pass. Notice that the number of free pages is still low — although they have been reclaimed, the pages are not yet free for reallocation.

**Example 11.4 Interval 4, Reclaiming Pages**

```
topaz 1> # Wait for reclaim task to wake up and run
topaz 1>  printit
System sleep: 60
%
topaz 1>  printit
System cacheStatistics:1
%
an Array
  #43 2021   [ReclaimedPagesCount]
  #76 178    [FreePages]
```

## Interval $I_5$ — Dispose of CR and Free Pages

Before the reclaimed pages can be added to the free pool, $CR_4$ must be disposed. SomeUser again provides the means by committing a change to the repository ($CR_5$). DataCurator again updates the session's transaction view by aborting the previous transaction. $CR_4$ can now be disposed. (Again, our example needs only a single transaction to reach this point; ordinarily, ongoing activity would supply the necessary commit records.)

Because all committed pages must be recorded for recovery purposes, the reclaimed pages will not show up in the free pool until after they have been saved as part of a checkpoint. For demonstration purposes, our example forces this action by sending the message `checkpoint` to System. The cache statistics now show 2293 free pages, or about 18.8 MBytes.

Notice that reclamation of these pages allowed the remaining possible dead objects to be promoted, now totalling 2440 (#87), and that 454 additional pages have been scheduled for reclamation (#91). The number of possible dead objects is now zero (#89).

**Example 11.5 Interval 5, Disposing of the CR and Freeing Space in the Repository**

```
topaz 1>  set session 2
topaz 2>  printit
      "Need another repository change for a new CR"
UserCommits := 3.
System commitTransaction
%
true
```

```
topaz 2>  set session 1
topaz 1> # Move this session to the new CR too.
topaz 1>  abort
topaz 1>  printit
SystemRepository cacheStatistics:1
%
an Array
  #76 178    [FreePages]
  #87 2440   [GcPromoteDeadCount]
  #89 0      [GcPossibleDeadSize]
  #91 454    [GcPagesNeedReclaiming]
topaz 1>  printit
      "If necessary, force a checkpoint to free pages"
System checkpoint
%
true
topaz 1>  printit
SystemRepository cacheStatistics:1
%
an Array
  #76 2293   [FreePages]
```

# Epoch Garbage Collection

Epoch garbage collection concentrates on identifying possible dead objects in a finite set of recent transactions, called the *epoch*. This approach is efficient for two reasons:

- The vast majority of objects die young, especially in applications characterized by numerous small transactions and where most of the transactions update a few previously committed objects.
- The cost of performing a transitive closure on the restricted set of recent transactions is much less than the cost of performing it on the entire repository.

The epoch algorithm makes use of the write set that GemStone maintains for each transaction. In terms of the transitive closure method described on page 11-7, the following definitions apply:

| | |
|---|---|
| RootSet | the set of all objects modified during the epoch |
| Universe | the set of all objects created during the epoch |
| PossibleDead | Universe minus Marked |

Although epoch garbage collection eliminates a large number of dead objects, it does not replace `markForCollection` for two reasons: First, objects that existed prior to the epoch are not considered, and hence dereferenced ones are not discovered. Second, if an object is created in one epoch and dereferenced in another, it will not be detected in either epoch.

Like `markForCollection`, epoch garbage collection only adds to the list of possible dead objects; it does not reclaim them itself.

For tuning information, see “To Tune Epoch Garbage Collection” on page 7-15.

## Performance Factors

The ability of epoch garbage collection to control growth of the repository by identifying dead objects depends on the relationship of three variables:

- the rate of production (R) of short-lived objects,
- the lifetime (L) of these objects, and
- the epoch length (E) between epoch garbage collection runs, for which the minimum is set by `#epochGcTimeLimit` in GcUser’s UserGlobals.

The Figure 11.8 shows the effect of the collection interval on the number of items marked as possibly dead. If L = E, for instance five minutes, each object is created in one epoch but survives until the next epoch (top part of figure). Because epoch

garbage collection detects only those objects that are created and die in the same epoch, none are collected.

When the collection interval is longer than an object's lifetime, however, some objects are created and die within the same epoch, and these can be collected. The lower part of Figure 11.8 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 collected. Only those created during the final third survive between epochs and are not collected.

The results shown in Figure 11.8 can be expressed as follows:

Objects Missed by EpochGC = $R \times L$

Objects Recovered by EpochGC = $R(E - L)$

where R is the rate of production of the short-lived objects of interest. For instance, assume R is 1000 objects per minute, L is 5 minutes, and E is 15 minutes. Then, for each collection interval,

Objects Missed = $1000 \times 5 = 5000$

Objects Recovered = $1000 (15 - 5) = 10000$

**Figure 11.8 Effect of Collection Interval on Epoch Garbage Collection**

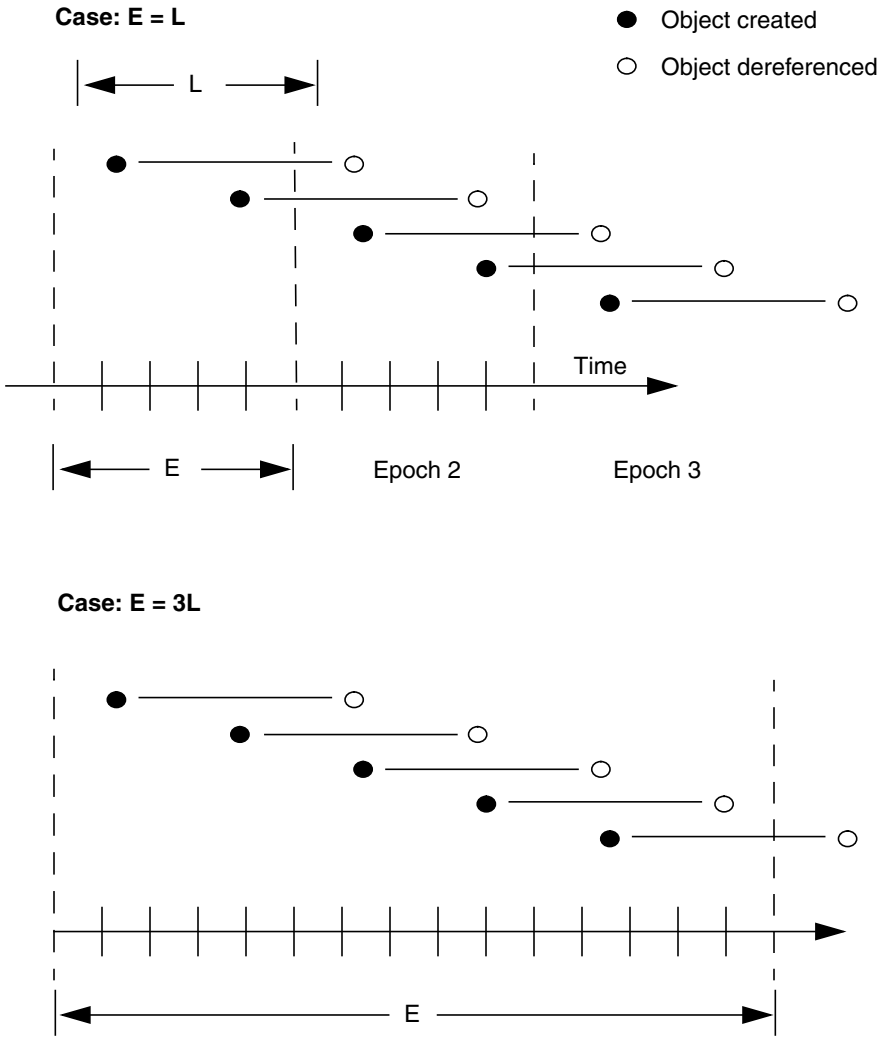


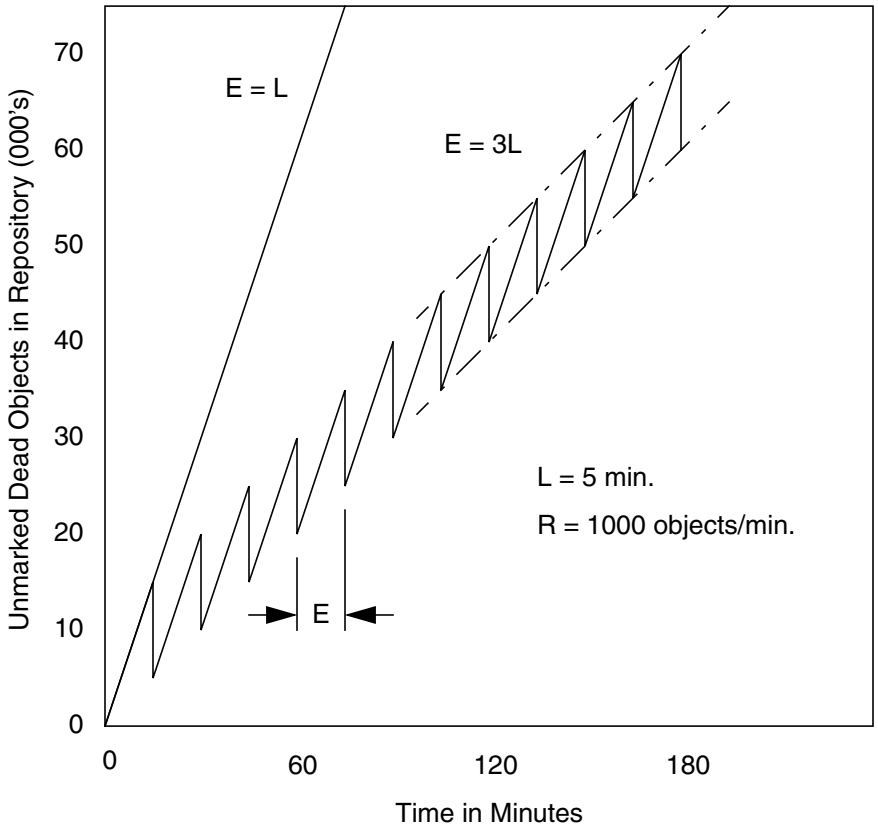
Figure 11.9 shows the effect of the collection interval graphically. This figure and the following one assume that the frequency of epoch garbage collection is determined by the minimum time interval (`#epochGcTimeLimit` ) because the byte and count thresholds always are met.

When E = L, in effect epoch garbage collection is disabled because all objects survive into the next epoch. As a result, the number of unmarked dead objects in the repository grows at the creation rate. These objects remain unmarked in the repository until `markForCollection` is run manually. This leads to the following rule:

Set `#epochGcTimeLimit` (E) > lifetime (L) of short-lived objects

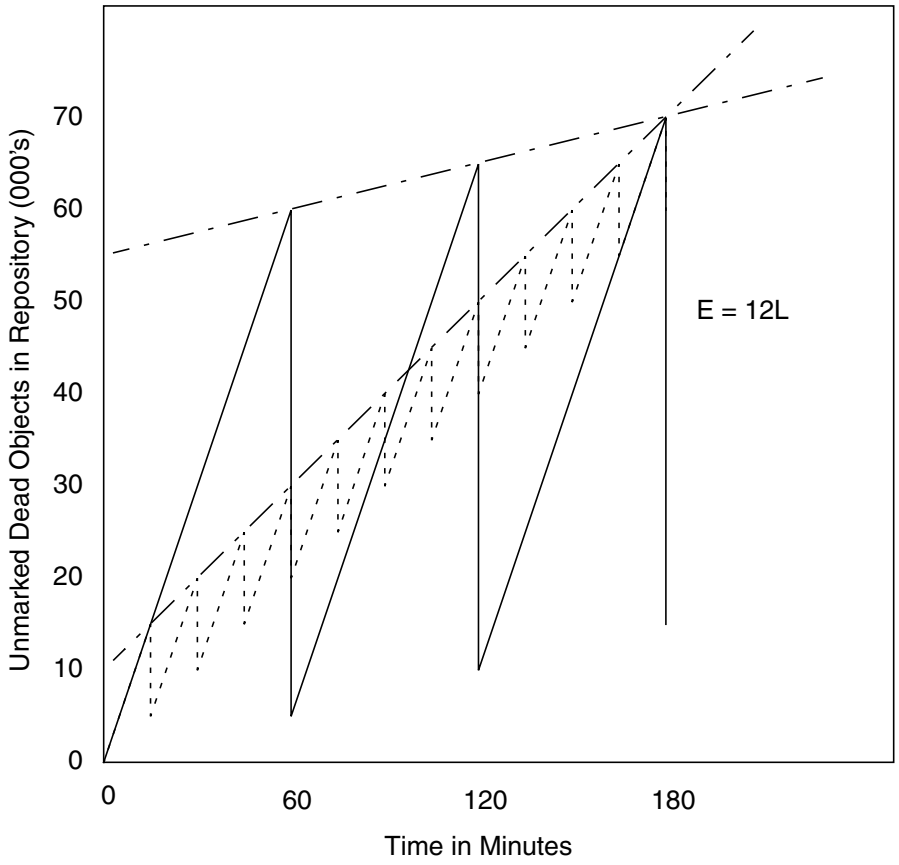
When the epoch collection interval is extended so that E = 3L, each epoch garbage collection marks those objects that were both created and dereferenced during that interval. This ratio causes the sawtooth pattern in the graph. If the creation rate is uniform, as in our example, two-thirds of the dead objects would be marked ((E–L) / E), and one-third would be missed (L / E). Consequently, the repository would grow at one-third the rate of the first case (where 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 control growth caused by those objects that survive between epochs.

**Figure 11.9 Repository Growth with Short Epoch**




Suppose we extend the collection interval to E = 12L. The result is shown in Figure 11.10 superimposed on part of the previous figure. Although the longer interval allows many more dead objects to accumulate, the growth rate of the repository is substantially less (25% of the previous case). The trade-offs are a need for greater head room on the disk at any given time and longer bursts of epoch garbage collection activity.

**Figure 11.10 Effect of Longer Epoch on Repository Growth**




## 11.5 Summary

This chapter has described the garbage collection process implemented in GemStone 5.0. The main points are these:

- GemStone collects two kinds of garbage: dereferenced objects, which are no longer referenced from symbol lists in AllUsers, and shadowed objects, which are previously committed objects that have been modified.
- Each Gem session process scavenges its local object memory on a generational basis as it needs room to create objects. Insufficient Gem temporary object

space can result in objects with short life expectancy being pushed to permanent object memory (disk).

- Epoch garbage collection scans recent transactions to mark dereferenced objects in permanent object memory as possibly dead.
- Applications can target specific objects for garbage collection by adding them to the global object GcCandidates. A GemStone administrator can invoke `markGcCandidates` to remove these objects if they are otherwise unreferenced.
- A GemStone administrator can invoke `markForCollection` periodically to scan the entire repository for dereferenced objects.
- An object is considered only possibly dead until each active session indicates by voting that it does not have a reference to that object in its Gem space.
- When a commit record is no longer needed, GcUser moves live objects to a new page, reclaims the identifiers of dead objects, and records the reclaimed page.
- The point at which resources become free depends on the transaction basis of other sessions and on the timing of checkpoints.

## 11.6 References

Knu69 Donald E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, chapter 2.3.5, pages 406-422. Addison-Wesley, Reading, Massachusetts, 1969.

Wils92 Paul R. Wilson. Uniprocessor Garbage Collection Techniques. In *International Workshop on Memory Management*, St. Malo, France, September 1992. Springer-Verlag Lecture Notes in Computer Science series.

GemStone

# Appendices

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** are files that 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** are files that 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, pageaudit, topaz,** and **gemsetup** accept command line arguments to point to configuration files:

- the **-z** option sets the system configuration file
- the **-e** option sets the executable-dependent configuration file

# 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. As shown in Figure A.1, GemStone first checks to see if there is an environment variable defined for GEMSTONE_SYS_CONF.

If GEMSTONE_SYS_CONF is *not* defined, GemStone looks for a file named *hostName*.conf in $GEMSTONE/data and uses that file. If no such file exists, it looks for a file named system.conf in $GEMSTONE/data 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.

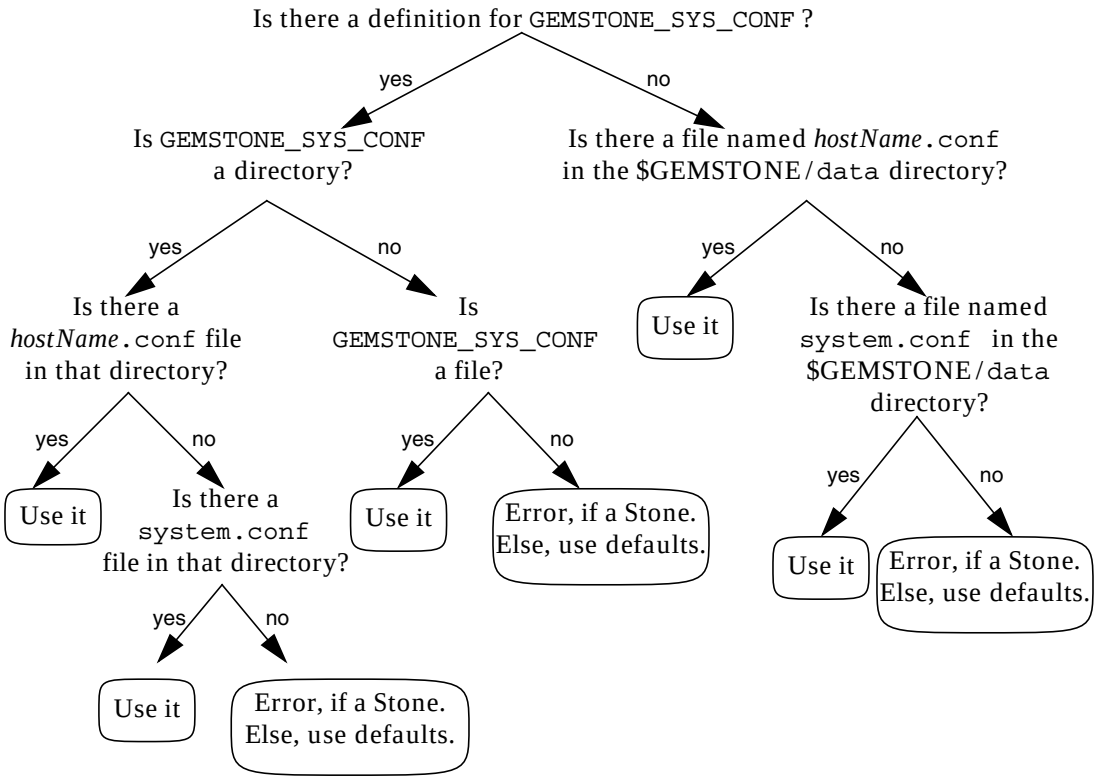
If GEMSTONE_SYS_CONF *is* defined, GemStone checks to see if it points to a directory.

- 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 an option is given a value in the system-wide configuration file and that option is also given a value in the executable-dependent file, the value in the executable-dependent configuration file overrides the system-wide configuration file's value.

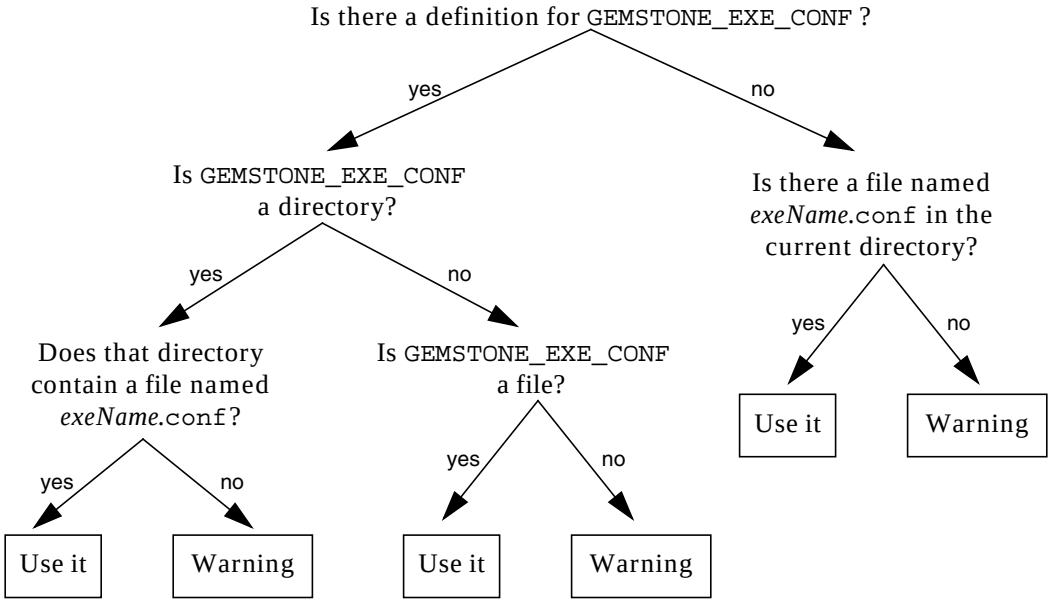
**Figure A.1 Search Path for a System-Wide Configuration File**



## Search for an Executable Configuration File

Ordinarily, GemStone repository executables next try to find an executable-dependent configuration file. (The exception is a Stone repository monitor that failed to find its system-wide configuration file—it exits with an error.) As shown in Figure A.2, GemStone begins by checking to see if there is an environment variable defined for GEMSTONE_EXE_CONF.

**Figure A.2 Search Path for an Executable Dependent Configuration File**




If no such environment variable is defined, GemStone tries to find a file called *exeName*.conf in the current working directory. (For information about the naming conventions, see "Naming Executable Configuration Files" on page A-6.) 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.

If there is an environment variable defined for GEMSTONE_EXE_CONF, 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 will look for a file named *exeName*`.conf` in the current working directory at start up. If it doesn't find one, it will use the configuration parameters set in the system-wide configuration file, or it will use 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 `$GEMSTONE/sys/gemne tobject` (Bourne shell) and `$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.

## 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 GemBuilder Applications

Linkable GemBuilder applications look for a file named `gci.conf` in the current working directory unless the application has provided a different name by calling GciInitAppName().

# Naming Conventions for Configuration Options

The prefix "GEM_" indicates that the option is processed directly by Gems. Unless indicated otherwise by the phrase "used by all executables," most other options are processed only by the Stone, which passes the information to executables as needed through network connections. Exceptions are the shared page cache configuration options ("SHR_"). The first Gem session process on a node remote from the Stone and extents reads these options, which determine the configuration of the shared page cache on that node.

All executables (that is, the Stone and Gems) understand the standard options used in the file $GEMSTONE/`data`/`system.conf` as shipped. The GemStone

executables 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 on page 1-38.*

## 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 (#).
Comments can be embedded in a configuration file using the following rules:

- by starting a line with the comment symbol
- by placing any text after the statement termination symbol (;)

**Lists** Lists are separated by commas; list elements can be empty, for example:

```
DBF_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 will read the configuration files. If any error is detected, then information about the error will be written to the standard output. This information includes which file and line contains the error and the error's severity.

There are two kinds of errors that can be generated by the processing of configuration files: syntax errors and option value errors.

## Syntax Errors

Syntax errors are generated whenever a grammatical error is detected in the configuration file. All syntax errors are warnings; they do not cause execution to terminate. These errors include:

- end-of-line or end-of-file detected before expected
- invalid starting character for an option name or invalid character within an option name
- equals or semicolon sign expected
- invalid 3-digit escape sequence
- invalid escape character
- terminating quote missing in a quoted string

### Option Value Errors

Option value errors are generated when the value assigned to an option has no meaning or is of the wrong type. For example, an option value error is generated when an option defined to need a boolean for its value has been set to an integer.

Option value errors vary in severity. Some options, such as not specifying the list of files that make up a logical repository, will necessarily terminate execution. Other option value errors, such as a invalid cache size, might only generate warnings. When a warning is issued, the executable will ignore the given value and use the option's default value.

## A.3 Configuration Options

The system-wide configuration file contains the following standard configuration options. *Default* in this discussion 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 1-39.

Note that there is a write-protected file called `initial.config` in the $GEMSTONE/`bin` directory 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 options are 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 list of all repository extents, in order, primary extent first, separated by commas. Taken together, all of the listed file resources make up the logical repository. This option is required, and must contain at least one entry, the name of the primary extent. The maximum number of extents is 255.

An extent name can be a file name or the device name for a raw disk partition. The name can have an environment variable as its first component. If an extent is on a remote host, the host name must be a network resource string; do NOT use an NFS

pathname or an environment variable that is set to an NFS pathname. If an extent is on a remote host, that host must support shared memory.

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 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. The sizes are in units of MBytes (1048576 bytes).

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.

For optimal performance using a raw partition, DBF_EXTENT_SIZES should be slightly smaller than the size of the partition so GemStone can avoid having to handle system errors. For example, set it to about 1995 MBytes for a 2 GByte 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; extents can grow until disk fills)
Minimum: 1 (Mbyte)
Maximum: operating system dependent

| Operating System | File System | Raw Partition |
|---|---|---|
| HPUX 10.0 | 2147 | 4095 |
| Solaris 2.4 | 2147 | 65535 |
| AIX 4.1 | 2147 | 65535 |
| Others | 2147 | at least 2147 |

## DBF_PRE_GROW

If DBF_PRE_GROW is set to True, then when a new extent is created, it is grown to its maximum size. If the new extent cannot be grown to the maximum size because of disk capacity problems, then the creation will fail.

The default value for DBF_PRE_GROW is False. This setting indicates that extents will grow only when new space is needed by the logical repository.

An extent without a maximum size is not pre-grown to any size; it is allocated a minimum size determined internally by the GemStone system.

DBF_PRE_GROW, if set to True, will affect existing extents at startup. If an existing extent has a maximum size and that extent is physically not at that maximum size, it is grown to that size and the added portion is initialized. If the grow fails, the extent is reset to its original size and the startup attempt fails.

Default: False

## DBF_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. If the replicate is on a remote host, the host name must be a network resource string and that host must support shared memory.

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. If the directory is on a remote host, the host name must be a network resource string.

Default: $GEMSTONE/`data`/

## DUMP_OPTIONS

If DUMP_OPTIONS is set to True, dumps a summary of all configuration options.

Default: True

## GEM_FREE_FRAME_LIMIT

Internal parameter: **#GemFreeFrameLimit**

When the number of free frames in the shared page cache is less than GEM_FREE_FRAME_LIMIT, the Gem session process scans the cache for a free frame rather than using one from the free frame list. This action is desirable for performance reasons so the remaining frames in the list are available for use by the Stone repository monitor.

Default: Set at login to 1/10 of the actual cache size, such as 125 frames for the default cache size of 10000 KBytes (1250 frames)
Minimum: 1
Maximum: 65536

## GEM_GCI_LOG_ENABLED

If GEM_GCI_LOG_ENABLED is set to True, Gci calls are logged to a separate GemBuilder transaction log owned by the Gem. This logging can seriously affect GemBuilder performance, and is provided for debugging referential integrity problems caused by bugs in language interfaces. These logs cannot be used for restore or recovery of transactions. They are only readable by a `pgsvr` process for debugging purposes.

This option has no effect in customer executables.

The following configuration parameters must be explicitly defined if GEM_GCI_LOG_ENABLED is True:

```
GEM_GCI_LOG_PREFIX = tranlogGci;
GEM_GCI_LOG_DIRECTORIES = .  /, . /;
GEM_GCI_LOG_SIZES = 100, 100;
```

Default: False

## GEM_HALT_ON_ERROR

GEM_HALT_ON_ERROR causes a Gem to halt and dump core if an error with the specified GemStone error number occurs. The value 0 means “never halt”. Ordinarily this option is used only to assist Technical Support in diagnosing problems.

Default: 0

## GEM_IO_LIMIT

Internal parameter: **#GemIOLimit**

GEM_IO_LIMIT limits the I/O rate to this number of I/Os per second (also applies to linked Gems). Values greater than 1000 result in the I/O rate being limited only by the performance of the underlying file system or disk partitions.

Default: 5000
Minimum: 1
Maximum: 65536

## GEM_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*
*Not all architectures 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*
*Not all architectures 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_PRIVATE_PAGE_CACHE_KB

GEM_PRIVATE_PAGE_CACHE_KB sets the size of the Gem's private page cache in KBytes. (This setting also applies to linked Gems.) This option expects a list of two values. The first value sets the size of the private cache when a shared page cache exists on the node where the Gem is running (in which case the Gem must use it). The second value sets the size of the private cache when a shared page cache is not available on the node where the Gem is running.

First value — size when a shared page cache exists:

Default: 200
Minimum: 64
Maximum: 65536

Second value — size when a shared page cache is NOT available:

Default: 6000
Minimum: 64
Maximum: 65536

## GEM_RPCGCI_TIMEOUT

GEM_RPCGCI_TIMEOUT specifies the time in minutes after which lack of an Rpc command will cause a Gem to terminate. Negative timeouts are not allowed. Resolution of timeouts is one half the specified time-out interval.

Default: 0 (Gem waits forever)
Minimum: 0

## GEM_SHR_PAGE_CACHE_ENABLED

GEM_SHR_PAGE_CACHE_ENABLED specifies whether a Gem should use a shared page cache. If the cache is enabled and a cache already exists on that system, that cache is used. If there is no shared cache on the system, one is created using the options SHR_PAGE_CACHE_SIZE_KB and SHR_PAGE_CACHE_NUM_PROCS.

If you set this option to False, be sure the second value of GEM_PRIVATE_PAGE_CACHE_KB is appropriate for your application. A minimum of 5000 KBytes is recommended.

Default: True

## GEM_TEMPOBJ_CACHE_SIZE

Internal parameter: **#GemTempObjCacheSize**.

GEM_TEMPOBJ_CACHE_SIZE sets the size of the Gem's temporary object space in KBytes. (This limit also applies to linked Topaz sessions and linked GemBuilder applications). The total memory allocation for managing temporary objects is (400 + GEM_TEMPOBJ_CACHE_SIZE)KBytes for GEM_TEMPOBJ_CACHE_SIZE ≥400, or (200 + GEM_TEMPOBJ_CACHE_SIZE)KBytes for GEM_TEMPOBJ_CACHE_SIZE <400.

If you increase the internal parameter at run time, for most efficient use of memory do it is as soon as possible after logging in, preferably soon enough that no garbage collection has occurred in the temporary object space. Attempts to set the size smaller than its current size generate an error.

Default: 600
Minimum: 200
Maximum: 10000

## LOG_WARNINGS

If LOG_WARNINGS is set to True, warnings are printed for invalid configuration options.

Default: True

## #NotConnectedDelta

The internal parameter `#NotConnectedDelta` sets the change in size of the notConnectedSet for a particular Gem that triggers another garbage collection. For information, see “Reclaiming the NotConnectedSet” on page 11-9.

Default: 300 objects

## #NotConnectedThreshold

The internal parameter `#NotConnectedThreshold` sets the minimum number of notConnected objects required to trigger the garbage collection. For information, see “Reclaiming the NotConnectedSet” on page 11-9.

Default: 2000 objects

## SHR_PAGE_CACHE_LOCKED

SHR_PAGE_CACHE_LOCKED specifies whether the shared page cache should be locked in memory. On systems that permit a portion of memory to be dedicated to GemStone, this option may provide higher performance. Specific operating systems may restrict this action to processes running as root or may require special privileges (such as MLOCK on HP-UX) for this option to take effect; for further information, check the shared page cache monitor log for error messages and consult your operating system documentation.

Default: False

## SHR_PAGE_CACHE_NUM_PROCS

SHR_PAGE_CACHE_NUM_PROCS sets the maximum number of processes allowed to attach to the shared page cache. The Stone uses one of this number on each cache; on the Stone’s node, the Stone’s AIO page server and the GC session each use one to attach to the cache. The process that starts the shared cache determines the maximum number of clients. The UNIX kernel must supply at least (SHR_PAGE_CACHE_NUM_PROCS + 1) semaphores.

The default (–1) causes this parameter to be derived from the STN_MAX_SESSIONS configuration option, using STN_MAX_SESSIONS + (number of extents) + 3.

Default: –1
Minimum: 15 or (number of extents + 3), whichever is larger, or –1 to use the default calculation.
Maximum: determined by STN_MAX_SESSIONS or kernel file descriptor limits.

## SHR_PAGE_CACHE_SIZE_KB

SHR_PAGE_CACHE_SIZE_KB sets the size of shared page cache in KBytes. (Additional shared memory is used for overhead.)

Default: 10000
Minimum: 512
Maximum: Limited by system memory and kernel configurations

## SHR_SPIN_LOCK_COUNT

Internal parameter: **#SpinLockCount**

SHR_SPIN_LOCK_COUNT specifies the number of tries to get a spin lock before the process sleeps on a semaphore. Semaphores involve a relatively time-consuming call to the operating system. Spin locks involve busy–wait loops. Efficient locking may require a combination of these methods.

In single processor architectures, this value should always be 1 since there is no value in spinning (the lock won't change until the process holding the lock gets scheduled). On multiple processor architectures, a value of 2000 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 2000 based upon the number of CPUs detected.

The internal parameter can be changed only by SystemUser.

Default: –1 (use either 1 or 2000, based on the number of CPUs detected)

## STN_CHECKPOINT_INTERVAL

Internal parameter: **#StnCheckpointInterval**

STN_CHECKPOINT_INTERVAL sets the maximum interval between checkpoints. Checkpoints may be written more often, depending on other factors. The unit is seconds.

The internal parameter can be changed only by SystemUser.

Default: 300
Minimum: 5
Maximum: 1800

## STN_DISABLE_LOGIN_FAILURE_LIMIT
## STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT

Internal parameters: **#StnDisableLoginFailureLimit**, **#StnDisableLoginFailureTimeLimit**

These options control when a user account is disabled because the user exceeded the STN_DISABLE_LOGIN_FAILURE_LIMIT of failed login attempts within the time in minutes specified by STN_DISABLE_LOGIN_FAILURE_TIME_LIMIT. When an account exceeds these limits, the user account is disabled (the system changes the password on the account to one that is invalid) and a record of the event is written to the Stone log file. The user account can only be restored by another user with OtherPassword privileges.

Changes to the internal parameters require the OtherPassword privilege.

STN_LOG_LOGIN_FAILURE_LIMIT:
Default: 15
Minimum: 0
Maximum: 65536

STN_LOG_LOGIN_FAILURE_TIME_LIMIT:
Default: 15
Minimum: 1
Maximum: 1440 (24 hours)

## STN_DISKFULL_TERMINATION_INTERVAL

Internal parameter: **#StnDiskFullTerminationInterval**

STN_DISKFULL_TERMINATION_INTERVAL specifies how soon (in minutes) the Stone should start terminating sessions holding on to the oldest commit record when the repository free space is below the value set for STN_FREE_SPACE_THRESHOLD. Such sessions are sent the fatal diskfull error.

The internal parameter can be changed only by SystemUser.

Default: 3
Minimum: 0 (no sessions are terminated)
Maximum: 1440 (24 hours)

## STN_FREE_SPACE_THRESHOLD

Internal parameter: **#StnFreeSpaceThreshold**

STN_FREE_SPACE_THRESHOLD sets the minimum amount of free space (in MBytes) 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 4-22.

The internal parameter can be changed only by SystemUser.

Default: 1
Minimum: 0 (no threshold)
Maximum: 65536

## STN_GC_SESSION_ENABLED

Internal parameter: **#GcSessionEnabled**

If STN_GC_SESSION_ENABLED is set to True, during start up the Stone creates a Gem process configured to reclaim unused space in disk pages and perform epoch garbage collection.

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 to release a commit record, after Stone has signaled that Gem that it should abort. If the time expires before the Gem aborts, the Gem will be forcibly aborted by Stone; forcible abort means the Gem will receive the error ABORT_ERR_LOST_OT_ROOT, and the Gem will have to completely reinitialize its object caches. Negative timeouts are not allowed. Resolution of timeouts is one half the specified time-out interval.

The internal parameter can be changed only by SystemUser.

Default: 1
Minimum: 1
Maximum: 1440

## STN_GEM_TIMEOUT

Internal parameter: **#StnGemTimeout**

STN_GEM_TIMEOUT sets the time in minutes after which lack of Gem interaction with the Stone will cause the Stone to terminate a session. Negative timeouts are not allowed. Resolution of timeouts is one half the specified time-out interval. If non-zero, this timeout is also the maximum time allowed for a Gem to complete processing of its login to the Stone. If this timeout is 0, the maximum time for login processing is set to five minutes.

The internal parameter can be changed only by SystemUser.

Default: 0 (Stone waits forever)
Minimum: 0

## STN_HALT_ON_FATAL_ERR

Internal parameter: **#StnHaltOnFatalErr**

If STN_HALT_ON_FATAL_ERR is set to True, the Stone will halt and dump core if it receives notification from a Gem that the Gem died with a fatal error that would cause Gem to dump core. By stopping the Stone at this point, the possibility of repository corruption is minimized. True is the recommended setting for systems during development.

If STN_HALT_ON_FATAL_ERR is set to False, the Stone will attempt to keep running if a Gem encounters a fatal error. False is the recommended setting for systems in production use.

The internal parameter can be changed only by SystemUser.

Default: True

Internal settings: 0 = False, 1 = True

## #StnLoginsSuspended

The internal parameter #StnLoginsSuspended ordinarily has the values 0 (False) and 1 (True) as set by `System | suspendLogins` and `resumeLogins`. Changing this parameter requires the SystemControl privilege.

## STN_LOG_LOGIN_FAILURE_LIMIT
## STN_LOG_LOGIN_FAILURE_TIME_LIMIT

Internal parameters: **#StnLogLoginFailureLimit**, **#StnLogLoginFailureTimeLimit**

If a user has greater than or equal to the STN_LOG_LOGIN_FAILURE_LIMIT number of login failures within the time in minutes specified by STN_LOG_LOGIN_FAILURE_TIME_LIMIT, a message is written to the Stone log file.

Changes to the internal parameters require the OtherPassword privilege.

STN_LOG_LOGIN_FAILURE_LIMIT:
Default: 10
Minimum: 0
Maximum: 65536

STN_LOG_LOGIN_FAILURE_TIME_LIMIT:
Default: 10
Minimum: 1
Maximum: 1440 (24 hours)

## STN_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 the number of users to be restricted to avoid overloading the host computer. The maximum number of file descriptors per process (imposed by the operating system kernel) can also limit the maximum number of sessions.

If you increase STN_MAX_SESSIONS 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: 2048

## 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 three causes each AIO page server to service $(4 + 2) \div 3 = 2$ extents.

A value greater than one is recommended only if there are two or more extents and you are trying to achieve the maximum possible commit rate. Values greater than four are unlikely to have any benefit.

Default: 1
Minimum: 1
Maximum: 30

## STN_PRIVATE_PAGE_CACHE_KB

STN_PRIVATE_PAGE_CACHE_KB sets the default size of Stone page cache in KBytes.

Default: 1000
Minimum: 64
Maximum: 65536

## 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.

The internal parameter can be changed only by SystemUser.

Default: 5
Minimum: 1
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 prefixes for transaction log file replicates, if any. A sequence number and "`.dbf`" is added to the prefix; for example, "`repltranlog`" produces "`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_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

## 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 8, "Managing Transaction Logs."

Default: None. The system will not run unless a value is provided.
Initial setting: False

## STN_TRAN_LOG_DEBUG_LEVEL

This option is only for GemStone internal use. Customers should not change the default setting. Values ≥ 2 inhibit removal of old transaction logs when in partial logging mode.

Default: 0

## STN_TRAN_LOG_DIRECTORIES

STN_TRAN_LOG_DIRECTORIES lists the directories or raw disk partitions to be used for transaction logging. This list defines the maximum number of log files that will be on line at once. Each entry must be a directory or a raw disk partition. Directories may appear multiple times in the list. A given raw disk partition may appear only once**.** If the directory is on a remote host, the host name must be a network resource string.

Default: Empty (the system will not run without at least two entries)
Minimum: 2 entries
Maximum: $2^{31}$ entries
Initial setting: $GEMSTONE/`data/`, $GEMSTONE/`data/`

## STN_TRAN_LOG_LIMIT

Internal parameter: **#StnTranLogLimit**

STN_TRAN_LOG_LIMIT sets the maximum transaction log entry size limit in KBytes. Successful commits of transactions consuming more than this amount of log file space when STN_TRAN_FULL_LOGGING is set to False will cause a checkpoint. This option has no effect when STN_TRAN_FULL_LOGGING is set to True.

The internal parameter can be changed only by SystemUser.

Default: 1000
Minimum: 25
Maximum: 1000

## STN_TRAN_LOG_PREFIX

STN_TRAN_LOG_PREFIX sets file name prefixes for transaction log files. A sequence number and "`.dbf`" is added to the prefix; for example, "`tranlog`" produces "`tranlog0.dbf`, `tranlog1.dbf` , ...". You can set this configuration option to permit multiple repository monitors to share a log directory without conflict.

Default: `tranlog`

## STN_TRAN_LOG_SIZES

STN_TRAN_LOG_SIZES sets the maximum sizes 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. The size is in MBytes, where one MByte is 1,048,576 bytes.

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 | configurationAt:.` These 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 description of `Repository | commitRestore` in the *GemStone Kernel Reference* manual.

## #SessionInBackup

`#SessionInBackup` is the GemStone session number of the session performing a full backup, or –1 if a backup is not in progress.

## #ShrPcTargetPercentDirty

`#ShrPcTargetPercentDirty` is used adjust how aggressive the Stone's AIO page servers are in writing repository pages to the disk.

## #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.

## #StnMntMaxAioRate

`#StnMntMaxAioRate` sets the maximum number of I/O operations per second for the Stone's AIO page servers. The effect is similar to that of #GemIoLimit.

## #StnRDbfMaxFilesPerServer

`#StnRDbfMaxFilesPerServer` is the approximate number of extents or replicated extents to be serviced by each of the Stone's AIO page servers. This value is calculated by dividing the total number of extents and replicated extents by the setting of the STN_NUM_LOCAL_AIO_SERVERS configuration option.

## #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
Usage: copydbf <sourceNRS> <destinationNRS> [ -     l | -  m | -  P ]
   [ -  f filePrefix ] [ -    n netLdiName ] [ -    p pgsvrId ]
   [ -  s Mbytes ] [ -    h ] [ -i | -I ]
copydbf sourceNRS -  i   [ - n netLdiName ] [ -   p pgsvrId ]
copydbf sourceNRS -  I   [ - n netLdiName ] [ -   p pgsvrId ]
```

UNIX man pages are available for more detailed information on each command.

# B.1 copydbf

**copydbf** *sourceNRS destinationNRS* [**-l** | **-m** | **-P**] [**-f***filePrefix*] [**-n***netLdiName*] [**-p***pgsvrId*] [**-s***Mbytes*] [**-h**]

**copydbf** *sourceNRS* **-i** [**-n***netLdiName*] [**-p***pgsvrId*]

**copydbf** *sourceNRS* **-I** [**-n***netLdiName*] [**-p***pgsvrId*]

*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.

**-l** least-significant-byte ordering. This ordering is the native byte ordering for Intel x86 processors, Sequent machines, and DECstations.

**-m** most-significant-byte ordering. This ordering is the default and is the native byte ordering for SPARC, IBM RS/6000, and HP PA-RISC (including HP9000/Series 700 and 800) machines.

**-P** preserve byte ordering. This option creates the destination file using the byte ordering found in the source file.

**-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 “netldi50”

**-p***pgsvrId* the name of a specific `runpgsvr` (similar to `gemnetid`)

| | |
|---|---|
| **-s***Mbytes* | The size to pre-allocate the destination file, in Mbytes. For instance, **-s**10 allocates at least 10 Mbytes 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. |

The **copydbf** utility requires exclusive access to repository files in order to copy them without corruption. You must shut down the Stone repository monitor before copying an extent. You may copy any transaction log that is not the active log.

GemStone repository files on the UNIX file system can usually be copied using the ordinary **cp** command. You can use the first form of **copydbf** for disk to disk copies between machines (without NFS), or copies 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 .
Source file: /users/GemStone5.0/data/extent0.dbf
   file type: extent fileId: 0
   Last checkpoint written at: Fri 07 Jun 1996 15:43:52 PDT.
Destination file: ./extent0.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** *sourceNRS* **-i**. 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:

```
% copydbf -i extent0.dbf
Source file: extent0.dbf
  file type: extent fileId: 0
  Last checkpoint written at: Mon 17 Jun 1996 11:07:54 PDT.
  Oldest tranlog needed for recovery is fileId 5
  ( tranlog5.dbf ).

% 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 17 Jun 1996
  11:21:20 PDT.
  Oldest tranlog needed for restore is fileId 5
  ( tranlog5.dbf ).
```

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** *sourceNRS* **-I**. This information is helpful in restoring a GemStone backup to a particular point in time. For example,

```
% copydbf -  I tranlog6.dbfa
Source file: tranlog6.dbfa
  file type: tranlog fileId: 6
  The file was created at:      Mon 17 Jun 1996 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: Mon 17 Jun 1996 10:56:29 PDT.
   oldest transaction references fileId 5 ( tranlog5.dbf ).
  Checkpoint 2 started at: Mon 17 Jun 1996 10:56:37 PDT.
   oldest transaction references fileId 5 ( tranlog5.dbf ).
  Checkpoint 3 started at: Mon 17 Jun 1996 11:07:49 PDT.
   oldest transaction references fileId 5 ( tranlog5.dbf ).
  Checkpoint 4 started at: Mon 17 Jun 1996 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 "/users/path/extent0.dbf_copy," via the standard GemStone network server "netldi50" 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.

```
% copydbf $GEMSTONE/bin/extent0.dbf /dev/rsd3h
```

Copying a transaction log from a raw partition to a file system directory generates a file name having the same form as if that transaction log had originated in a file system. If the internal fileId is 43, this example would name the destination file `/dsk1/tranlogs/tranlog43.dbf:`

```
% copydbf /dev/rsd3h /dsk1/tranlogs/
```

# 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 |
| killed | server 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 NT, 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 krbsrv

**krbsrv** *operation* [**-h**]

The **krbsrv** utility lets the Kerberos administrator create an authentication database, add or change entries, and generate secrets files for hosts.

*operation* must be one of the following (**-h** displays a usage line and exits):

**init** [**-n**] *realmName* [*databaseName*]
Create a new Kerberos database on this host for the realm *realmName*. If **-n** is not specified, **init** prompts for the master key, which is used to encrypt every key stored in the database.

**-n** causes a stashed master key to be fetched from a cache file.

*databaseName*
name of the Kerberos database file; defaults to `/kerberos/principal` .

**edit** [**-n**] Edit principals in the Kerberos database. When executed without **-n**, **edit** prompts for the master key string and verifies that it matches the master key stored in the database.

Once the master key has been verified, edit begins a prompt loop. The user is prompted for the principal and instance to be modified. If the entry is not found, the user can create it. Once an entry is found or created, the user can set the password, expiration date, maximum ticket lifetime, and attributes. Default expiration dates, maximum ticket lifetimes, and attributes are presented in brackets; if the user presses return the default is selected. There is no default password. The password RANDOM is interpreted specially, and if entered the user may have the program select a random key for the principal.

Upon successfully creating or changing the entry, ‘‘Edit O.K.’’ is printed. To exit from the loop, enter a blank principal name.

**-n** causes a stashed master key to be fetched from a cache file.

**list** [**-n**] List the contents of a Kerberos database in a text representation. When executed without **-n**, **list** prompts for the master key string and verifies that it matches the master key stored in the database.

**-n** causes a stashed master key to be fetched from a cache file.

**stash** Stash the database master key in a cache file, `/.k`. The user is prompted to enter the key, to verify the authenticity of the key and the authorization to store the key in the file.

**server** [**-s**] [**-m**] [**-n**] [**-p** *pauseSeconds*] **[-a** *maxAge*] [**-l** *logFile*] [**-r** *realm*] [*databasePathname*]
Start the daemon used to authenticate one principal to another.

**-s** sets defaults for a slave server.

**-m** prompts the user for the master key; otherwise, the stashed key is read from its cache.

**-n** allows the user to specify an infinite maximum age for the database (only useful with **-s** and overrides **-a**).

**-p** *pauseSeconds*
sets the delay between an unrecoverable error and exiting to *pauseSeconds* (300 to 3600); otherwise, if **-s** is not used, the daemon will pause forever.

**-a** *maxAge*
sets the maximum age in seconds to *maxAge* (3600 to 259200), above which the database is considered too old for a slave server to use; the default is infinite.

**-l** *logFile*
sets the name of the file for messages to *logFile*; the default is `/kerberos.log`.

**-r** *realm* sets the name of the *realm* for which the server will give information; the default is the local realm.

*databasePathname*
name of the Kerberos database file; defaults to `/kerberos/principal`.

**srvtab** [**-n**] [**-r** *realm*] *instance* [*instance* ...]
Create a `srvtab` on this machine for a given realm and instances. When executed without **-n**, **srvtab** prompts for the master key string and verifies that it matches the master key stored in the database.

For each *instance* (host name*)* specified on the command line, **srvtab** creates the service key file *instance*`-new-srvtab`, containing all the entries in the database with an instance field of *instance*. This new file

contains all the keys registered for Kerberos-mediated server programs, such as `netldi50`, to run on the host *instance*.

**-n** causes a stashed master key to be fetched from a cache file.

**-r** *realm*
realm fields in the extracted file will match the given *realm* rather than the local realm.

*instance*
name of the host for which the `srvtab` is to be prepared.

**onesecret** [**-n**] *principal.instance* [*principal.instance* ...]
Create a `srvtab` but only for the given principals. The file created is `secrets.new` in the current directory. When executed without **-n**, **onesecret** prompts for the master key string and verifies that it matches the master key stored in the database. For example, you can use this option to extract a `srvtab` that only includes `netldi50`.

**-n** causes a stashed master key to be fetched from a cache file.

*principal.instance*
name of a Kerberos user or service on a particular machine.

**destroy** After prompting for confirmation, destroy the Kerberos database on this machine.

# B.4 krbtool

**krbtool** *operation* [**-h**]

The **krbtool** utility lets users log in to the Kerberos system and obtain an initial ticket, list their tickets, and destroy them.

**-h** displays a usage line and exits.

**init** [**-i**] [**-r**] [**-v**] [**-l**] [*name*]
Log in to the Kerberos system and obtain an initial ticket (note that only registered users may log in). When you use **init** without options, the utility prompts for your username and Kerberos password, and tries to authenticate your login with the local Kerberos server. If Kerberos authenticates the login attempt, the utility retrieves your initial ticket and puts it in the ticket file specified by your KRBTKFILE environment variable. If this variable is undefined, your ticket will be stored in the directory specified by the TMPDIR environment variable, if defined, or else in the `/tmp` directory. The default file name is `tkt`*uid*, where *uid* specifies your user identification number.

**-i** prompts for an instance.

**-r** prompts for a realm, which lets you authenticate yourself with a remote Kerberos server. (This option is not fully implemented in Kerberos Version 4.)

**-v** causes the utility to use verbose mode, which prints the realm and a status message indicating success or failure.

**-l** prompts for a ticket lifetime in minutes (5 to 1275).

*name* Kerberos attempts to log in user *name* and prompts for *name*'s Kerberos password.

**list** [**-s** | **-t**] [**-file** *filename*] [**-srvtab**]
Print the name of the tickets file and the identity of the principal the tickets are for (as listed in the tickets file). List the principal names of all Kerberos tickets currently held by the user, along with the issue and expire time for each authenticator.

**-s** suppresses printing of the issue and expire times, the name of the tickets file, and the identity of the principal.

**-t** checks for the existence of a non-expired ticket-granting-ticket in the ticket file; if one is present, the utility exits with status 0, else it exits with status 1 (no output is generated when this option is specified).

**-file** *filename*
takes the following argument as the ticket file; otherwise, if the KRBTKFILE environment variable is set, it is used. If this variable is undefined, the directory specified by the TMPDIR environment variable is used, if defined, or else the directory `/tmp`. The default file name is `tkt`*uid*, where *uid* specifies your user identification number.

**-srvtab**
the file is treated as a service key file, and the names of the keys contained therein are printed (you must have read privilege for the file). If no file is specified with a **-file** option, the default is `srvtab` in the current directory.

**destroy** [**-f**] [**-q**]
Destroy all active tickets in the user's ticket file; if the ticket file does not exist, display a message to that effect. This operation is intended to be used before system log out to prevent others from using unexpired tickets; the command can be placed in a `.logout` file.

**-f** the status message is not displayed.

**-q** the utility does not beep your terminal when it fails to destroy the tickets.

# B.5 pageaudit

**pageaudit** [*gemStoneName*] [-**e***exeConfig*] [-**z***systemConfig*] [-**h**]

| | |
|---|---|
| *gemStoneName* | name of the GemStone repository monitor; the default is "gemserver50". Network resource syntax is not permitted. |
| -**e***exeConfig* | the GemStone executable-dependent configuration file |
| -**z***systemConfig* | the GemStone system configuration file |
| -**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 A-2 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.

This utility can take a long time to run, so it is best to run it as a background job.

For additional information about **pageaudit** and a description of its output, see "To Perform a Page Audit" on page 7-29.

# B.6 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 "netldi50" |
| **-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, transaction log, or backup file 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 7-8.

The options for this command generally are not needed for a standard GemStone configuration.

# B.7 startnetldi

**startnetldi** [*netLdiName*] [**-l***logFile*] [**-t***timeout*] [**-a***name*] [**-k**] [**-n**] [**-g** | **-s**] [**-d**]

*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 "netldi50".

**-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 120 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.

**-k** kerberos only; no username/password authentication shall be allowed. The file `/etc/srvtab` must be readable and contain an appropriate principal entry.

**-n** no *ad hoc* processes shall be created (ad hoc processes are ones not listed in `$GEMSTONE/bin/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

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 or Kerberos 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 3-9.

For assistance with start-up failures, refer to “To Troubleshoot NetLDI Startup Failures” on page 4-9.

# B.8 startstone

**startstone** [*gemStoneName*] [**-l***logFile*] [**-e***exeConfig*] [**-z***systemConfig*] [**-h**] [**-N**] **[-R]**

*gemStoneName* name of the GemStone repository monitor, default is "gemserver50". 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 4-3.

# B.9 stopnetldi

**stopnetldi** [*netLdiName*] [**-h**]

*netLdiName* the name of the GemStone network server; the default is "netldi50". 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.10 stopstone

**stopstone** [*gemStoneName* [*gemStoneUserName* [*gemStonePassword*] ] ] [**-i**] [**-h**]

*gemStoneName* the name of the GemStone repository monitor, commonly "gemserver50". 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. For assistance, refer to "To Troubleshoot Session Login Failures" on page 4-14.

# B.11 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) "netldi50" |
| **-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 "How GemStone Uses Configuration Files" on page A-2 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.12 waitstone

**waitstone** [*gemStoneName* | *netLdiName*] [*timeout*]

*gemStoneName* the name of the GemStone repository monitor, commonly `gemserver50`

*netLdiName* the name of the GemStone network server, commonly `netldi50`

*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 "gemserver50" running on node "handel", you would specify two NRS strings:

```
topaz> set gemstone !@handel!gemserver50
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\!gemserver50 -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!gemserver50
% waitstone "!tcp@oboe#auth user@password!gemserver50"
```

## C.2 Defaults

The following items uniquely identify a network resource:

- communications protocol— such as TCP/IP, DECnet, or SNA
- destination node—the host that has the resource
- authentication of the user—such as a system authorization code
- resource type—such as server, database extent, or task
- environment—such as a NetLDI, a directory, or the name of a log file

- resource name—the name of the specific resource being requested.

A network resource string can include some or all of this information. In most cases, you need not fill in all of the fields in a network resource string. The information required depends upon the nature of the utility being executed and the task to be accomplished. Most GemStone utilities provide some context-sensitive defaults. For example, the Topaz interface prefixes the name of a Stone process with the **#server** resource identifier.

When a utility needs a value for which it does not have a built-in default, it relies on the system-wide defaults described in the syntax productions in "Syntax" on page C-4. You can supply your own default values for NRS modifiers by defining an environment variable named GEMSTONE_NRS_ALL in the form of the *nrs-header* production described in the Syntax section. If GEMSTONE_NRS_ALL defines a value for the desired field, that value is used in place of the system default. (There can be no meaningful default value for "resource name.")

A GemStone utility picks up the value of GEMSTONE_NRS_ALL as it is defined when the utility is started. Subsequent changes to the environment variable are not reflected in the behavior of an already-running utility.

When a client utility submits a request to a NetLDI, the utility uses its own defaults and those gleaned from its environment to build the NRS. After the NRS is submitted to it, the NetLDI then applies additional defaults if needed. Values submitted by the client utility take precedence over those provided by the NetLDI.

## 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** | **decnet** | **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** | **decnet** | **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*] | **#krb** )

**#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.

**#krb** specifies that kerberos authentication is to be used instead of a user name and password.

*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 `netldi50`. (See the earlier discussion of *nrs-identifier*.)

**#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:

| | |
|---|---|
| %H | home directory |
| %M | machine's network node name |
| %N | executable's base name |
| %P | process pid |
| %U | user name |
| %% | % |

*resource-modifier* ::= ( **#server** | **#spawn | #task** | **#dbf** | **#monitor** | **#file** )

Identifies the intended purpose of the string in the *nrs-body*. An NRS can contain only one resource modifier. The default resource modifier is context sensitive. For instance, if the system expects an NRS for a database file, then the default is **#dbf**.

**#server** directs the NetLDI to search for the network address of a server, such as a Stone or another NetLDI. If successful, it returns the address. The *nrs-body* is a network server name. A successful lookup means only that the service has been defined; it does not indicate whether the service is currently running. A new process will not be started. (Authorization is needed only if the NetLDI is on a remote node and is running in secure mode.)

**#task** starts a new Gem. The *nrs-body* is a NetLDI service name (such as "gemnetobject"), followed by arguments to the command line. The NetLDI creates the named service by looking first for an entry in `$GEMSTONE/bin/services.dat` , and then in the user's home directory for an executable having that name. The NetLDI returns the network address of the service. (Authorization is needed to create a new process unless the NetLDI is in guest mode.) The **#task** resource modifier is also used internally to create page servers.

**#dbf** is used to access a database file. The *nrs-body* is the file spec of a GemStone database file. The NetLDI creates a page server on the given node to access the database and returns the network address of the page server. (Authorization is needed unless the NetLDI is in guest mode).

**#spawn** is used internally to start the garbage-collection 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 three elements: **SystemUser**, **DataCurator**, and **GcUser**.

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.)

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.

## 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 lists the contents of this dictionary. For information about the kernel classes, see the *GemStone Kernel Reference* manual. 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.

*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 operations except those that* ***require*** *SystemUser privileges, such as upgrade and full restores.*

**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 |
| **Collections** | #AllClusterBuckets | ClusterBucketArray |
| | #AllGroups | CanonicalStringDictionary |
| | #AllSymbols | CanonicalStringDictionary |
| | #AllUsers | UserProfileSet |
| | #ConfigurationParameterDict | SymbolKeyValueDictionary |
| | #ErrorSymbols | SymbolDictionary |
| | #GemStoneError | SymbolDictionary |
| | #Globals | SymbolDictionary |
| | #InstancesDisallowed | IdentitySet |

**Table D.1 Initial Contents of the Globals Dictionary (Continued)**

| | Key | The object's class |
|---|---|---|
| **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** | | |

Appendix

# E GemStone Directory Contents

This appendix lists the files that are included in your GemStone system.

## E.1 GemStone Directory

### $GEMSTONE

The GemStone directory is created at installation with a path name of the form:

*InstallDir*/`GemStone5.0-` *platform*

where:

| | |
|---|---|
| *InstallDir* | is your installation directory, and |
| *platform* | is a suffix indicating the architecture and operating system of the host machine, for example “sparc.Solaris” for a Sun SPARCstation or “hppa.hpux” for an HP workstation. |

This directory is assigned to the environment variable GEMSTONE. Other directories can be referenced from this variable. For example, the default directory containing your repository is `$GEMSTONE/data`.

Your $GEMSTONE directory contains the following files:

| | |
|---|---|
| PACKING | a list of all GemStone files |
| version.txt | information identifying this release of GemStone |

This directory also contains 13 subdirectories in which reside the GemStone files. These subdirectories are:

| | | | |
|---|---|---|---|
| bin | examples | lib | sys |
| data | include | patches | ualib |
| doc | install | pub | upgrade |

The following sections list the contents of each of these directories.

# E.2 GemStone Bin Directory

## $GEMSTONE/bin

This directory contains the following files:

| | |
|---|---|
| bash | the Gnu version of the Bourne shell (sh), public domain software from the Free Software Foundation. bash also incorporates useful features from the Korn and C shells (ksh and csh). It is ultimately intended to be a faithful implementation of the IEEE Posix Shell and Tools specification (IEEE Working Group 1003.2). |
| copydbf | an executable file that copies GemStone DBF extents, replicates, transaction logs, and backup files. |
| extent0.dbf | the initial GemStone repository file, containing Smalltalk kernel classes and methods. Provided here only for archival purposes. |
| gemsetup.csh | a C shell source script that is used to select the GemStone version. |
| gemsetup.sh | a Bourne shell dot script that is used to select the GemStone version. |
| ggetopt | a version of getopt(1), a utility for parsing shell arguments to be used by other shells. This version is based on the Gnu getopt(3) function. |
| gslist | an executable file to help a GemStone system administrator examine the state of GemStone server processes. |

| | |
|---|---|
| `initial.config` | the default system-wide configuration file. Provided for archival purposes only. |
| `krbsrv` | GemStone tool for Kerberos server administration. |
| `krbtool` | GemStone tool for Kerberos users. |
| `misc.sh` | a Bourne shell script that sets up various useful functions and defaults for other scripts. |
| `nrspec.sh` | an internal shell script for parsing network resource strings. |
| `pageaudit` | a shell script used to do page-level audits of a repository file. |
| `removedbf` | a utility used to remove a GemStone repository extents, transaction logs, and full backup files from a raw disk partition. |
| `saymessage` | an executable file used to access the GemStone language-independent message system. |
| `services.dat` | a text file used by `netldi`; a network services database that correlates network service names with executable files on this host. |
| `setconfig.csh` | an internal C shell source script that sets GemStone configuration parameters. |
| `setconfig.sh` | an internal Bourne shell dot script that sets GemStone configuration parameters. |
| `startnetldi` | an executable file that starts a GemStone network server process in the background. |
| `startstone` | an executable file that starts a GemStone repository monitor process in the background. See Appendix B for more information about the startstone command. |
| `statmonitor` | an executable for use by GemStone consultants or under direction of Technical Support. |
| `stonelist` | an obsolete shell script to help a GemStone system administrator examine the state of GemStone server processes. This script has been replaced by the `gslist` executable and will be removed in the next release. |
| `stopnetldi` | an executable that stops the GemStone network server process. |

| | |
|---|---|
| `stopstone` | a shell script that conducts an orderly shutdown of the GemStone repository monitor. (See Appendix B for more information about the stopstone command.) |
| `topaz` | executable file for Topaz. For more information, see the *GemStone Topaz Programming Environment*. |
| `waitstone` | an executable file that is used to test whether the GemStone repository monitor or GemStone network server process is on line and ready to accept commands. |
| `woman` | a portable replacement for the `man` utility. |
| `wwhatis` | a version of `whatis` that works with `woman`. |

# E.3 GemStone Data Directory

## $GEMSTONE/data

This directory contains the following files.

| | |
|---|---|
| `extent0.dbf` | the repository file (optionally copied from the GemStone bin directory during installation). |
| `system.conf` | the default system-wide configuration file. |

# E.4 GemStone Documentation Directory

## $GEMSTONE/doc

The documentation directory contains the files:

| | |
|---|---|
| `bugnotes.txt` | a list of bugs known to exist in this release |
| `errormessages.txt` | a list of error messages by number in two groups: messages stored in the repository, and messages stored in compiled form in `$GEMSTONE/sys/english50.err` . |
| `fixman` | a script called by `makewhatis`. |
| `hierarchy.txt` | a report of the class hierarchy for this release. |
| `makewhatis` | a script that makes GemStone man page whatis database. |

Three subdirectories contain man pages for the various GemStone utility programs:

```
man1        man5        man8
```

# E.5 GemStone Examples Directory

## $GEMSTONE/examples

The contents of the examples directory are organized in two subdirectories, `admin` and `smalltalk`.

The `admin` directory contains three sample configuration files and an example script for archiving transaction logs:

| | |
|---|---|
| `archivelog.sh` | an example script showing how to archive transaction logs out of raw partitions. It must be modified to reflect the actual locations of your raw partitions. |
| `large.conf` | an example configuration file for 200 users, a 10 GByte repository, and 1 GByte of RAM. It uses five extent files. |
| `medium.conf` | an example configuration file for 50 users and a 1 Gbyte repository, and 256 MBytes of RAM. It uses one extent file. |
| `small.conf` | an example configuration file for 12 users, a 100 Mbyte repository, and 64 Mbytes of RAM. It uses one extent file. |

The `smalltalk` directory contains several unsupported samples of general interest. The file `goodies.gs` may be of particular interest; it contains sample Smalltalk expressions that allow you to perform useful GemStone system functions.

# E.6 GemStone Include File Directory

## $GEMSTONE/include

This directory contains files for use with GemBuilder for C Interface functions. For more information, see the *GemBuilder for C* manual.

These files are intended to be included directly by programmers:

| | |
|---|---|
| `gci.hf` | an include file for C code that binds at build time to a GemBuilder for C library. |
| `gcifloat.hf` | an include file defining C functions to swizzle binary floats; used with code that binds at build time to a GemBuilder for C library. |
| `gcirtl.hf` | an include file for C code that binds at run time to a GemBuilder for C library |
| `gciua.hf` | an include file for C code that makes up a user action library |
| `staticua.hf` | an include file for C code that defines static user actions |

The following files are used by one or more of the files in the first group and do not need to be included directly:

| | |
|---|---|
| `flag.ht` | contains host-specific GemBuilder definitions for use in C programs. |
| `gci.ht` | defines C types for use by GemBuilder functions. |
| `gcicmn.ht` | defines GemBuilder for C types and macros used in common |
| `gcierr.ht` | defines mnemonics for all GemStone errors. (See the *GemBuilder for C* manual for information about error mnemonics.) |
| `gcioc.ht` | defines C mnemonics for sizes and offsets into objects. |
| `gcioop.ht` | defines C mnemonics for predefined GemStone objects. |
| `gcirtl.ht` | defines types used by `gcirtl.hf` |
| `gcirtlm.hf` | defines macros used by `gcirtl.hf` |
| `gciuser.hf` | defines macros used by `gciua.hf` |
| `version.ht` | defines C mnemonics for version-dependent strings. |

# E.7 GemStone Install Directory

## $GEMSTONE/install

This directory is used for the installation procedure. It contains the following files:

`Makefile` a Makefile for public domain programs used by GemStone.

`asyncio, asyncio.c`
a utility to verify that the kernel configuration includes asynchronous I/O.

`buildsystests` a script to rebuild executables that are used during installation and which also can be useful later for support purposes.

`checksum` checks the GemStone tree against the `PACKING` file to make sure that all the necessary files are present.

`checksums.good` result file for use with `checksum`.

`fileopentest, fileopentest.c`
a utility to test how many files can be simultaneously opened by a process.

`getdtablesize, getdtablesize.c`
a utility that returns the maximum number of file descriptors, using the `getdtablesize(2)` call.

`gethostbyname, gethostbyname.c`
a utility that looks up the net address of a specified host.

`getid, getid.c` a utility that prints the real user ID, real group ID, effective user ID, or effective group ID of the current process.

`getopt.c` the Gnu version of getopt(), a utility for parsing shell arguments to be used by other shells. This version allows flexible argument order.

`getpwnam, getpwnam.c`
a utility that tests the getpwnam() system call, which accesses the operating system user database.

`getservbyname, getservbyname.c,`
a utility that verify presence of given services by name.

`getservbyport, getservbyport.c`
a utility that verify presence of given services by port number.

| | |
|---|---|
| `gnugetopt.h` | previously named getopt.h, this is the interface to the Gnu getopt() function. |
| `installgs` | an automated script for installing GemStone. |
| `machine_grp` | a script that creates any missing GemStone installation subdirectories, and sets the group, user name, and default protection on all GemStone directories and files. |
| `makeusers` | an automated script for changing default system passwords and adding GemStone users once GemStone has been installed. |
| `shmem, shmem.c` | a utility to verify that the kernel shared memory configuration will support a given shared page cache size and number of sessions. |
| `snefru` | the Xerox Secure Hash Function, a one-way hash function that provides authentication. `snefru` is freeware. |

# E.8 GemStone Library Function Directory

## $GEMSTONE/lib

The GemStone library functions are in this directory. It contains the following files. For more information, see the *GemBuilder for C* manual.

| | |
|---|---|
| `gcirtlobj.o, gcirtlobj_p.o` | object file used when binding GemBuilder for C at run time; `gcirtlobj_p.o` exists only on platforms that require shared library code to be compiled position independent. |
| `gciualib.o` | object file used when building a user action library |
| `gemrpcobj.o` | object file used when statically linking user action code (C functions that can be called from Smalltalk) into a Gem |
| `libgcilnk50.so` | the shared library for the GciLnk (linkable) version of the GemBuilder for C. |
| `libgcirpc50.so` | the shared library for the GciRpc (remote procedure call) version of the GemBuilder for C. |
| `topazobj.o` | object file used when statically linking user action code into Topaz |

# E.9 GemStone Patches Directory

## $GEMSTONE/patches

This directory provides a location for installing patches to this release.

# E.10 GemStone Pub Directory

## $GEMSTONE/pub

This directory contains published software from sources outside of GemStone.

`bash.license` a text file stating the conditions of the Gnu public license for bash.

`bash-1.14.tar.Z` The complete bash distribution, in accordance with the bash license agreement.

`woman.tar.Z` sources for `woman`.

`snefru.tar.Z` sources for `snefru`.

`woman.license` license file for the `woman` program.

## $GEMSTONE/pub/krb

This directory contains Kerberos components. The U.S. Government prohibits export of some software used by Kerberos. See the Makefile in this directory for further information. The file `kerberos.FAQ` contains answers to a set of Frequently Asked Questions; it was posted to the news group comp.protocols.kerberos.

| | | | |
|---|---|---|---|
| `Makefile` | `krb.c` | `krbdes.c` | `krbsrv.c` |
| `getopt.c` | `krb.hf` | `krbdes.hf` | `krbsu.c` |
| `gnugetopt.h` | `krbdb.c` | `krbpriv.hf` | `krbtool.c` |
| `kerberos.FAQ` | `krbdb.hf` | | |

# E.11 GemStone Sys Directory

## $GEMSTONE/sys

This directory contains the following files:

| | |
|---|---|
| `english50.err` | a data file of compiled GemStone messages written in English (the default). |
| `filedesc.csh` | C shell script that sets file descriptor limit to be used when running GemStone programs. GemStone administrators can modify this file (wizards only). |
| `gem` | the executable image that implements a GemStone RPC session. |
| `gemnetobjcsh` | C shell script invoked by `netldi` to start a GemStone session process. |
| `gemnetobject` | Bourne shell script invoked by `netldi` to start a GemStone session process. |
| `gemstone.key` | a "key file" used by GemStone's copy protection mechanism. Ordinarily you create this file during installation from information shipped with the GemStone distribution tape. If you do not have that information and the file is not present, call GemStone Contract Administration. |
| `msgcom` | a utility to compile GemStone message text into a data file like `enlish.err` or to convert a data file back into a text source file of like that described in Appendix G. |
| `netldid` | a GemStone network server process designed to overcome the limitations of `/etc/services` and NIS. |
| `pgsvr` | a version of `pgsvrmain` that contains debugging information for use by GemStone engineers. |
| `pgsvr.hlp` | pgsvr online help file. |
| `pgsvrmain` | the executable image that implements a GemStone disk demon. It is invoked by the `runpgsvr` script, which in turn is invoked by `netldi`. |
| `pgsvrslow` | a version of `pgsvrmain` that should be used only on instructions from GemStone Customer Support. |
| `rungc` | Bourne shell script to start the Garbage Collection session. |

| | |
|---|---|
| `runpgsvr` | Bourne shell script invoked by `netldi` in response to a request of the `netldi` to provide GemStone repository I/O. |
| `runpgsvrcsh` | a C shell script invoked by `netldi` in response to a request of the netldi to provide GemStone repository I/O. |
| `runpgsvrmain` | Bourne shell script invoked by `netldi` in response to a request of the `netldi` to provide GemStone repository I/O. |
| `runpgsvrmaincsh` | a C shell script invoked by `netldi` in response to a request of the netldi to provide GemStone repository I/O. |
| `shrpcmon.hlp` | shared page cache monitor online help file. |
| `shrpcmonitor` | the executable image that implements the shared page cache monitor. |
| `startshrpcmonitor` | Bourne shell script that `netldi` uses to start the shared page cache monitor. |
| `statmonitor.hlp` | statmonitor help file. |
| `stoned` | the executable file for the Stone (GemStone repository monitor) process. This process controls and provides access to the GemStone repository file. |
| `topaz.hlp` | Topaz online help file. |

# E.12 GemStone User Action Libraries

## $GEMSTONE/ualib

This directory is intended to hold user action libraries for applications. It is empty in the initial installation.

# E.13 GemStone Upgrade Directory

## $GEMSTONE/upgrade

The upgrade directory contains a number of Smalltalk kernel class definitions in files with `.gs` suffixes. The file `sizedbag.gs` contains Smalltalk code illustrating how to subclass btree and sort nodes to improve the performance of equality indexes on user-defined classes.

In addition, the directory contains subsidiary upgrade scripts with the `.topaz` suffix.

Appendix

# F 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.

## F.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/GemStone5.0-hppa.hpux` .

GEMSTONE_CHILD_LOG
: Used by parent process to tell child process the name of its log file. To keep the child's log from being deleted when the process terminates normally, unset this variable in the appropriate script, such as `$GEMSTONE/sys/gemnetobject` .

GEMSTONE_EXE_CONF
: The location of an executable-dependent configuration file; see "Creating an Executable Configuration File" on page A-5.

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 G-1.

GEMSTONE_LOG
: The location of system log files for the Stone repository monitor and its child processes. For further information, see “GemStone Server Logs” on page 10-2.

GEMSTONE_MAX_FD
: Limits the number of file descriptors requested by a GemStone process. For further information, see “Estimating File Descriptor Needs” on page 1-12.

GEMSTONE_NRS_ALL
: Sets a number of network-related defaults, including the type of user authentication that GemStone expects. For further information, see “To Set a Default NRS” on page 3-15.

GEMSTONE_SYS_CONF
: Location of a system-wide configuration file; see “How GemStone Uses Configuration Files” on page A-2.

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_DISABLE_KEEPALIVE
: A non-empty string disables the network keepalive facility. For further information about keepalive, see “Disrupted Communications” on page 3-8.

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`.

KRB_ETC
: The location of Kerberos files that would ordinarily be in `/etc`. For information, see “The Non-Root Server Option” on page H-4.

KRB_ADMIN
: The location of Kerberos files that would ordinarily be in `/kerberos`. For information, see “The Non-Root Server Option” on page H-4.

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.

KRBTKFILE
: The location in which to store Kerberos ticket granting tickets; see“To Obtain a Ticket” on page H-12.

PATH
: The search path of locating executable files.

SHELL
: Used to determine what shell to use for an exec, such as by `System | performOnServer`.

TERM
: Used by Kerberos.

# F.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

# G Translation Files for Messages

GemStone prints error and advisory messages by inserting arguments into text it extracts from a language-dependent, pre-compiled error file. GemStone currently provides only one such file, `$GEMSTONE/sys/english50.err` . This appendix explains how create and compile the source of that file and how users 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*.

## G.1 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*`50.err`. If GEMSTONE_LANG is not defined, the default value $GEMSTONE/`sys/english50.err` is used.

## G.2 The Message Compiler

GemStone messages are stored in a pre-compiled error file, $GEMSTONE/sys/*language*50.err, for efficient retrieval at run time. Users who want to translate the default english50.err in to another language can regenerate the language source by running the **msgcom** utility to produce a text file. After translation, the new language source can be compiled 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 G.1**

```
% $GEMSTONE/sys/msgcom totext $GEMSTONE/sys/english50.err \
english.lang
...
            wrote 187044 characters
```

## G.3 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 G.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, and
- the resulting messages for sample arguments.

**Example G.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/Gemstone5.0/data/system.conf
     GEMSTONE_EXE_CONF=/user2/application.conf
startstone[Info]: checking if server gemserver50 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 maximum length of a 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 mapping line. The syntax of a mapping line is:

   <*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 ten 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 G.2, the first error symbol is "`startstone.info`" and there are two arguments, which are the configuration file paths.

Example G.2 also illustrates the GemStone practice 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. 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, and so forth.

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, and |
| 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, with a few exceptions that will be noted. 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”.

The exceptions mentioned above are that the translation of arguments is “smarter” than `printf()` in these ways:

- 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 versus int versus 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. Here are some examples followed by 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.

## G.4 Creating New Message Files

This section discusses the subtleties, pitfalls, and stylistic points in writing error messages.

First and most importantly, the symbol itself must NOT be translated. 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 G-2.

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

There are a few categories of messages in the system that don't translate to other languages:

- GemStone Copyright Notice—This copyright notice has its text imbedded 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 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....).

Appendix

# H Kerberos

This appendix tells you how to configure Kerberos to work with GemStone and explains the commands available to users. Kerberos is an authentication service for open networks that was developed by Project Athena at MIT. The GemStone distribution tape contains Kerberos executables built from Release 4 sources.

*NOTE*

*GemStone is aware of CERT Advisory CA-96.03 and has determined that the use of Kerberos in this release is not affected.*

Kerberos at its core consists of a client database, an authentication server, and protocols for logging in users and mutually authenticating users and network services. In the case of GemStone, NetLDIs can use Kerberos to authenticate a user before creating a GemStone process. Because Kerberos maintains private encrypted keys (passwords) for each *principal*, it provides a more secure alternative to the use of `.netrc` files for identifying trusted users and services. The design is such that unencrypted passwords are never sent over the network or stored on servers.

## H.1 Installing Kerberos

There are two ways to install Kerberos for use with GemStone:

- If Kerberos is already is use at your site, add the NetLDI principal with an instance for each host on which it will be run. See the instructions under "How to Add GemStone to an Existing Kerberos System," which follows.
- If you are installing Kerberos for the first time, you can either use the Project Athena distribution, following their instructions, or you can use the executables in the $GEMSTONE directory and follow the instructions under "How to Install a New Kerberos System."

## How to Add GemStone to an Existing Kerberos System

If your site is already running Kerberos and you want to use it with GemStone, execute the following steps after your usual Kerberos setup procedure for the hosts involved. These instructions assume you are already familiar with administration of Kerberos as it is installed at your site.

**Step 1.** Use **kdb_edit** (or **kadmin**) to register "netldi50" as a principal, and create an instance for each host on which a NetLDI will be run.

**Step 2.** Use **ext_srvtab** to create the file *hostName*`-new-srvtab` for each host in Step 1. Copy that file to `/etc/srvtab` on the corresponding host, and change its protection to 400.

## How to Install a New Kerberos System

This procedure tells you how to install a Kerberos system by using executables in the GemStone distribution. The basic steps, in outline form first, are these:

1. Select the server machine, update the network entries, and create the Kerberos configuration files.
2. Create the initial Kerberos database.
3. Add an initial principal (yourself) to the database.
4. Start the Kerberos daemon.
5. Test the installation thus far.
6. Add the NetLDI principal with an instance for each host.
7. Install a configuration file on each host and synchronize their clocks.
8. Create and install a key file for each host that will run a NetLDI.
9. Register other users as necessary.

10. Verify that `$GEMSTONE/sys/netldid` is owned by root and has the setuid bit set.

## The krbsrv Utility

The following procedure uses commands of the form **krbsrv** *operation*. The **krbsrv** executable is located in `$GEMSTONE/bin` and packages several administrative commands from the Project Athena release in a way that avoids ambiguity if both are in your UNIX path. Table H.1 shows the mapping between the Project Athena executables and the commands provided by the GemStone package. The Project Athena Version 4 commands and their GemStone equivalents can be used interchangeably if both executables are in your UNIX path. Only the GemStone utilities recognize the environment variables KRB_ADMIN and KRB_ETC, which are described under "The Non-Root Server Option," next.

**TABLE H.1 Kerberos Administrative Commands**

| Project Athena | GemStone Name | Purpose |
|---|---|---|
| **kdb_init** | **krbsrv init** | Create and initialize a master database |
| **kdb_edit** | **krbsrv edit** | Create or change principals (users and servers) stored in the master database |
| — | **krbsrv list** | List the current database entry for each principal in text form |
| **kdb_destroy** | **krbsrv destroy** | Destroy a master database |
| **kstash** | **krbsrv stash** | Stash the database master key in a hidden file that initially is readable only by root |
| **kerberos** | **krbsrv server** | Start the Kerberos master server daemon |
| **ext_srvtab** | **krbsrv srvtab** | Extract a `srvtab` for all registered services on a particular host. |
| **—** | **krbsrv onesecret** | Extracts a `srvtab` only for the services specified. |

## The Non-Root Server Option

Ordinarily, Kerberos is installed in the /kerberos and /etc directories, and the authentication server process is owned by root. However, the Kerberos executables distributed with GemStone allow nonprivileged users to run the server out of their own directories. To run the server this way, create two directories and set the environment variables KRB_ADMIN and KRB_ETC to point to them. For example:

```
% cd $HOME
% mkdir krb_admin krb_etc
```

Then set the environment variables to point to the directories you created (C shell):

```
% setenv KRB_ADMIN $HOME/krb_admin
% setenv KRB_ETC $HOME/krb_etc
```

or (Bourne shell),

```
$ KRB_ADMIN=$HOME/krb_admin
$ KRB_ETC=$HOME/krb_etc
$ export KRB_ADMIN KRB_ETC
```

Both **krbsrv** and **krbtool** (the GemStone packages for administration and user utilities, respectively) will use those directories in place of /kerberos and /etc. The directories can be shared among workstations. Most parts of the following installation procedure can be done by an ordinary user who has defined KRB_ADMIN and KRB_EDIT. The exception is the first part of Step 1, which edits /etc/hosts and /etc/services; that part ordinarily must be done by root.

## To Install the Kerberos System

**Step 1.** One machine in your network will run the Kerberos authentication server daemon, and we will take the name of that machine as your Kerberos *realm*, or authentication domain. In the example that follows, that machine is "oboe."

- ❒ Log in to the server machine as root.
- ❒ Edit the /etc/hosts file (or the NIS master) to give that machine the alias "kerberos." For example,

  ```
  192.83.233.27 oboe kerberos
  ```

- ❒ Edit the /etc/services file (or the NIS master) to add the port entries like the following. which are the ports recommended by Project Athena:

```
kerberos  750/tcp  kdc  # Kerberos authentication--tcp
kerberos_master  751/tcp  # Kerberos authentication
kerberos_master  751/udp  # Kerberos authentication
kerberos  750/udp  kdc  # Kerberos authentication--udp
```

You can use other port numbers less than 1024 or greater than 5000 that aren't being used by other services. Some operating systems are already configured for Kerberos, so check for existing entries.

*NOTE:*
*If the port numbers are less than 1024, the Kerberos server must be owned by root. If the numbers are greater than 5000, the server can be owned by any user.*

- ❒ If you are installing a non-root server (page H-4), log out as root and perform the remain steps as that nonprivileged user.
- ❒ Create the file `/etc/krb.conf` (or `$KRB_ETC/krb.conf`), which specifies the system's local realm on the first line. Subsequent lines specify a realm and the name of the machine on which the Kerberos server runs for that realm. The file has this form:

  *localRealmName*
  *realmName masterServerName*

  For example (where *realmName* is taken from *masterServerName*):

  ```
  oboe
  oboe oboe
  ```

**Step 2.** Create the initial Kerberos database.

- ❒ Make sure that `$GEMSTONE/bin` is in root's path.
- ❒ Create a Kerberos directory and make it writable only by root. (For non-root installations, this directory is `$KRB_ADMIN`.)

  ```
  # mkdir /kerberos
  # chmod 755 /kerberos
  ```

- ❒ Create the initial database, specifying the realm. You will be prompted for a master password for the Kerberos system. The command line is

  **`krbsrv init`** *realmName* [*kerberosDatabaseName*]

For example:

```
# krbsrv init oboe
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter Kerberos master key:  hush
```

Ordinarily, the database files are created in `/kerberos`. For non-root installations, they are created in `$KRB_ADMIN`.

**Step 3.** Add an initial principal to the database, then add the GemStone NetLDI as a principal.

❒ Add yourself to the database as a *principal*. Where defaults are provided in [brackets], you can accept them by pressing the Return key. (The passwords you register with Kerberos have no connection with UNIX passwords and, to maintain security, they should be different.)

*NOTE:*
*Many Kerberos utilities will not work properly unless the Kerberos principal is the same as that user's login name.*

```
# krbsrv edit
Opening database...
Enter Kerberos master key: hush
Current Kerberos master key version is 1.
Master key entered. BEWARE!
Previous or default values are in [brackets] ,
enter return to leave the same, or new value.
Principal name: myName
Instance:       <press Return>
<Not found>, Create [y] ?
Principal: YourName, Instance: , kdc_key_ver: 1
New Password:  myKerberosPassword
Verifying, please re-enter
New Password:  myKerberosPassword

Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [ 1999-12-31 ] ?
Max ticket lifetime (*5 minutes) [ 255 ] ?
Attributes [ 0 ] ?      <not used; press Return>
Edit O.K.
Principal name:         <continued below>
```

❒ Add the NetLDI name "netldi50" to the Kerberos database as a principal with one instance for each host on which a NetLDI will be run.

*NOTE:*
*Register "netldi50" as the principal in the following step even if you choose to start the NetLDI process under a different name.*

Continue **krbsrv edit** to create an instance for each host. Press the Return key to accept default values. Entering "RANDOM" as the password for a server causes Kerberos to generate a random password that will be known only to the master server and that particular server instance. For example, to create NetLDI instances for hosts *oboe* and *flute*:

```
Principal name:  netldi50
Instance: oboe
<Not found>, Create [y] ?
Principal: netldi50, Instance: oboe, kdc_key_ver: 1
New Password:  RANDOM
Verifying, please re-enter
New Password:  RANDOM
Random password [y] ?
Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [ 1999-12-31 ] ?
Max ticket lifetime (*5 minutes) [ 255 ] ?
Attributes [ 0 ] ?
Edit O.K.
Principal name:  netldi50
Instance: flute
<Not found>, Create [y] ?
Principal: netldi50, Instance: flute, kdc_key_ver: 1
New Password:  RANDOM
Verifying, please re-enter
New Password:  RANDOM
Random password [y] ?
Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [ 1999-12-31 ] ?
Max ticket lifetime (*5 minutes) [ 255 ] ?
Attributes [ 0 ] ?
Edit O.K.
Principal name:       <press Return to end the edit>
```

**Step 4.** Start the Kerberos daemon. If desired, prepare the daemon for automatic restarting.

- Store the master key (in effect) in file `/.k` (or `$KRB_ETC/.k` for the non-root server) by invoking this command:

```
# krbsrv stash
Enter Kerberos master key:  hush
Current Kerberos master key version is 1.
Master key entered. BEWARE!
```

- Start the daemon as a background process. If the server is to run as root (which includes all servers with a port number less than 1024), you must start it while logged in as root. The stored master key will be read from the cache file. (If you did not store the master key, use **krbsrv server -m** instead of the command line in the example.)

```
# krbsrv server &
Kerberos server starting
 Sleep forever on error
 Log file is /kerberos/kerberos.log
Current Kerberos master key version is 1.
Master key entered. BEWARE!
Current Kerberos master key version is 1
Local realm: oboe
```

- Add the preceding command to a UNIX start-up file, typically `/etc/rc.local`, so that the Kerberos daemon will start automatically when the machine is booted.

**Step 5.** Test the installation thus far. Log in as yourself (the user name that you added to the Kerberos database). You should be able to obtain a Kerberos

*ticket-granting ticket* (tgt), list your current tickets, and then destroy the tickets. For example:

```
% krbtool init
MIT Project Athena (oboe)
Kerberos Initialization
Kerberos name: myName
Password: myKerberosPassword
% krbtool list
Ticket file: /tmp/tkt234
Principal: myName@oboe
IssuedExpiresPrincipal
Jan 20 13:32:47  Jan 20 21:32:47  krbtgt.oboe@oboe
% krbtool destroy
Tickets destroyed.
% krbtool list
Ticket file: /tmp/tkt234
list: No ticket file (tf_util)
```

**Step 6.** Set up the other hosts in your realm.

- Copy `/etc/krb.conf` (or `$KRB_ETC/krb.conf`) to each host in your realm where users will need to obtain tickets and to each host that will run a NetLDI.
- Make sure that the system clocks are at least loosely synchronized (within a few minutes).

**Step 7.** Create and install a key file (`/etc/srvtab` or `$KRB_ETC/srvtab`) for each host that will run a NetLDI. There are two ways to do this: You can use **krbsrv srvtab**, which creates a file containing secrets for *all* Kerberos services on a given host, or you can use **krbsrv onesecret**, which contains only the secrets for specific services, such as `netldi50.oboe`.

- To create the comprehensive file, execute **krbsrv srvtab** *hostName*. The file *hostName*`-new-srvtab` will be created in the current directory. That file

contains encrypted keys to authenticate all server instances on that host, much the way a password typed at the keyboard authenticates a user. For example:

```
# krbsrv srvtab oboe -n
Current Kerberos master key version is 1.
Master key entered. BEWARE!
Generating 'oboe-new-srvtab'....
# krbsrv srvtab flute -n
Current Kerberos master key version is 1.
Master key entered. BEWARE!
Generating 'flute-new-srvtab'....
```

The **-n** option tells Kerberos to read the stored master key.

Move each *hostName*`-new-srvtab` file to the `/etc` directory on *hostName* (or to `$KRB_ETC/srvtab` for non-root installations) and rename it `srvtab`. For root servers, the file must be owned by root and have protection 400. For non-root servers, it must be readable by the user who runs the server.

❒ To create the limited file, execute **krbsrv onesecret** *principal.instance* ... . The file `secrets.new` will be created in the current directory. That file contains encrypted keys to authenticate the specific server instances you requested (you can enter multiple names). For example:

```
# krbsrv onesecret -n netldi50.oboe
Current Kerberos master key version is 1.
Master key entered. BEWARE!
Writing file secrets.new
```

Move each `secrets.new` to `/etc/srvtab` or `$KRB_ETC/srvtab` on the host (instance) where it will be used before generating another secrets file.

**Step 8.** Register other GemStone users as principals, following the procedure in Step 3.

**Step 9.** For servers that run as root, verify that `$GEMSTONE/sys/netldid` is owned by root and has the setuid bit set (UNIX permission 6555). This step is necessary so that the NetLDI can read the server key file, `/etc/srvtab`, which is readable only by root. If you are using the non-root installation, `netldi` must be owned by a user who has permission to read `$KRB_ETC/srvtab`. A generic Kerberos failure message can have various causes, but a likely one is inability to read the server key file.

# H.2 How to Use Kerberos

This section tells GemStone users how to use the Kerberos system for network authentication. Examples in Chapter 3, "Connecting Distributed Systems," give details for a number of typical GemStone configurations.

Before you can use Kerberos, the administrator must have added you to the authentication database and assigned a password. Ordinarily, your Kerberos user name will be the same as your UNIX user name. However, your Kerberos password likely will be different (UNIX and Kerberos passwords are independent).

## Kerberos Names

A Kerberos name contains three parts:

1. The first is the *principal* name, which is usually the name of a user or a service, such as GemStone's NetLDI service.
2. The second is the *instance*, which typically is null for a user. In the case of a service, the instance is the name of the machine on which it runs. When writing a Kerberos name, the principal name is separated from the instance (if it is not null) by a period; for example, `netldi50.oboe` is the an instance of the GemStone 5.0 NetLDI server that runs on a host named oboe.
3. The third part of a Kerberos name is the *realm*, or authentication domain, which identifies the specific Kerberos service that provides authentication for the principal. The realm might take its name from an Internet domain or from the name of the host on which the Kerberos server runs. When writing a Kerberos name, separate the principal and instance from the realm by an @ sign; for example, `netldi50.oboe@servio`.

## Kerberos Tickets

Kerberos authenticates users by issuing them *tickets*, which are encrypted messages that can be understood only by the master server and a particular server instance, such as `netldi50.oboe`. When you successfully log in to the Kerberos system, you are granted an initial *ticket-granting-ticket* (tgt), which lets you obtain individual tickets for specific services later in your session. Only issuance of the initial ticket is visible to you as a user; the rest is done transparently.

The procedures in this section use commands of the form **krbtool** *operation*. The **krbtool** executable is located in `$GEMSTONE/bin` and packages several user commands from the Project Athena release in a way that avoids ambiguity if both

are in your UNIX path. Table H.2 shows the mapping between the Project Athena executables and the commands provided by GemStone. The Project Athena Version 4 commands and their GemStone equivalents can be used interchangeably.

**TABLE H.2 Kerberos User Commands**

| Project Athena | GemStone Name | Purpose |
|---|---|---|
| **kinit** | **krbtool init** | Logs a registered user in to the Kerberos system and obtains an initial ticket |
| **klist** | **krbtool list** | Lists the tickets currently held by a user |
| **kdestroy** | **krbtool destroy** | Destroys a user's active tickets; used before logging out |

## To Obtain a Ticket

If you will be using the non-root installation described earlier (see "The Non-Root Server Option" on page H-4), be sure that you have defined the environment variables KRB_ETC and KRB_ADMIN.

Use the command **krbtool init** to log in to the Kerberos system and obtain an initial ticket. When you use **krbtool init** without options, the utility prompts for your user name and Kerberos password, and tries to authenticate your login with the local Kerberos server. For example:

```
% krbtool init
MIT Project Athena (oboe)
Kerberos Initialization
Kerberos name: Mary
Password: mysecret
```

If Kerberos authenticates the login attempt, the login utility retrieves your initial ticket and puts it in the file specified by the environment variable KRBTKFILE. If that variable is undefined, the file is placed in the directory specified by the TMPDIR environment variable, if defined, or else in `/tmp`. The default file name `/tkt`*uid*, where *uid* is your UNIX user id.

If Kerberos responds with a message like the following, you haven't been registered as a Kerberos user or Kerberos has not been installed on that host. See your Kerberos administrator.

```
init: Principal unknown (kerberos)
```

## To List Your Tickets

Use the command **krbtool list** to print your active tickets. The utility prints the name of the ticket file, who the tickets are for, and the principal (a service) for each ticket. In the following example, the first ticket is for a special ticket-granting service and was obtained by **krbtool init**. The second ticket, for `netldi50` service, was obtained on the user's behalf using the ticket-granting ticket when the user requested Topaz RPC to start a Gem session. The user did not have to give her Kerberos password again nor is it stored as text.

```
% krbtool list
Ticket file: /tmp/tkt234
Principal: Mary@oboe
Issued  Expires  Principal
Jan 24 15:34:00  Jan 24 23:34:00  krbtgt.oboe@oboe
Jan 24 15:34:2  Jan 24 23:34:28  netldi50.oboe@oboe
```

Each ticket has a lifetime that is fixed when it is issued (eight hours is typical). If your ticket expires, you can obtain a new initial ticket by repeating **krbtool init**. You can request a different ticket lifetime by using the **-l** option (see the description of **krbtool init** on page B-12).

## To Destroy Your Tickets

To protect system security, you should destroy any active tickets before you log out. You can destroy them by using **krbtool destroy**, which you may want to put in a UNIX `.logout` file. Destroying a ticket ensures that someone else cannot log in to your workstation and find a way to use the ticket. The Kerberos dialog is simple:

```
% krbtool destroy
Tickets destroyed.
```

# Index

## Symbols



## A

## B

## C

## D

## E

## F



## G

## I



## K

## L



## M

## N

## O



## P

## R

## S

## T

# U

## V



## W