GemStone®

GemStone/S 64 Bit Release Notes

Version 2.2.1

May 2007

GemStone S 64™

INTELLECTUAL PROPERTY OWNERSHIP

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

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

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

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

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

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

COPYRIGHTS

PATENTS

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

TRADEMARKS

GemStone, **GemBuilder**, **GemConnect**, and the GemStone logos are trademarks or registered trademarks of GemStone Systems, Inc. in the United States and other countries.

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

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

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

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

Microsoft, **MS**, **Windows**, **Windows 2000** and **Windows XP** are registered trademarks of Microsoft Corporation in the United States and other countries.

Linux is a registered trademark of Linus Torvalds and others.

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

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

Other company or product names mentioned herein may be trademarks or registered trademarks of their respective owners. Trademark specifications are subject to change without notice. All terms mentioned in this documentation that are known to be trademarks or service marks have been appropriately capitalized to the best of our knowledge; however, GemStone cannot attest to the accuracy of all trademark information. Use of a term in this documentation should not be regarded as affecting the validity of any trademark or service mark.

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

Preface

About This Documentation

These release notes describe changes in the GemStone/S 64 Bit version 2.2.1 release. We recommend that everyone migrating to this version read these release notes before beginning installation, testing or development.

No separate Installation Guide is provided with this release. For instructions on installing GemStone/S 64 Bit version 2.2.1, or upgrading or converting from previous products or versions, see the Installation Guide for version 2.2.

These documents are also available on the GemStone customer website, as described below.

Terminology Conventions

This document uses the following terminology:

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

Technical Support

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

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

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

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

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

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

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

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

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

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

Community provides customer forums for discussion of GemStone product issues.

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

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

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

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

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

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

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

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

Email: support@gemstone.com

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

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

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

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

24x7 Emergency Technical Support

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

Training and Consulting

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

- Training courses are offered periodically at GemStone's offices in Beaverton, Oregon, or you can arrange for onsite training at your desired location.
- Customized consulting services can help you make the best use of GemStone products in your business environment.

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

Contents

Chapter 1. GemStone/S 64 Bit 2.2.1 Release Notes

Overview 1-1
Changes and New Features 1-1
Keyfile changes 1-1
Added tranlog analysis abilities 1-1
Number of application write locks increased to 10 1-1
Explicit session termination timeout 1-2
New errors 1-2
Topaz changes 1-2
IFERR. 1-2
IFERR_LIST 1-3
IFERR_CLEAR 1-3
DISPLAY / OMIT ERRORCHECK. 1-3
Changes to IFERROR 1-3
Bugs Fixed 1-3
Possible hang in Gem processes 1-3
Unable to start stone on Linux without network 1-3
Integer raisedTo: returned Float 1-4
Locale not used in ScaledDecimal read / writes 1-4
Some time zones encountered failures 1-4
Poor performance in Fraction comparison. 1-4
Some complier errors reported as ObjErrBadOffset 1-4
RcQueue>>removeObject:addedBySessionId: incorrect behavior 1-4
Could not use slow stone executable in fast installation 1-4
Topaz died on login error 1-4
Problems with very large OOPs 1-4

Chapter 2. Object State Change Tracking

Introduction 2-1
Tranlog Analysis Scripts 2-2
Script Prerequisites 2-2
Output 2-2
Tranlog Assumptions 2-2
Filter Criteria. 2-3
printlogs.sh. 2-4
Examples 2-4
searchlogs.sh 2-5
Examples 2-5
Tranlog Structure 2-6
Tranlog Entries 2-7
Tranlog Entry Types 2-7
Very Large Objects 2-8
Full vs. Normal Mode. 2-9
Example of Tranlog Analysis 2-11
Tracking Changes to an Employee 2-11
Changed vs. new objects 2-14
Details of Changes to an Employee 2-14
Further Analysis 2-17
Class Operations. 2-17
Deleted Objects 2-17
Managing Volume. 2-17

Chapter 1

GemStone/S 64 Bit 2.2.1 Release Notes

Overview

GemStone/S 64 Bit 2.2.1 is a new version of the GemStone/S 64 Bit object server. This release provides a number of new features, improved performance, and fixes a number of bugs; we recommend everyone using or intending to upgrade to GemStone/S 64 Bit 2.2, upgrade to this new version. The details of these changes are provided in this document.

These release notes provide changes between the previous version of GemStone/S 64 Bit, version 2.2, and version 2.2.1. If you are upgrading from a version prior to 2.2, please also review the release notes for each intermediate release to see the full set of changes.

No separate Installation Guide is provided with this release. For installation instructions, use the Installation Guide for version 2.2.

Changes and New Features

Keyfile changes

New parameters, CPU affinity and GCI Traversal, have been added to the keyfiles. As a result, keyfiles for version 2.2 will not work for version 2.2.1.

Added tranlog analysis abilities

This release adds a feature that allows analysis of a set of tranlogs to be used to determine the history of changes to application objects. For details on this, see Chapter 2, ”Object State Change Tracking”.

Number of application write locks increased to 10

Version 2.2 introduced application write locks, providing two application write lock queues. The number of application writes lock queues has been increased to 10 in version 2.2.1.

The cache statistics Lock1WaitQueueSize and Lock2WaitQueueSize have been removed; they are replaced by the new cache statistics LockWaitQueueSize1 through LockWaitQueueSize10.

Explicit session termination timeout

To bypass the normal `stopSession:` timeout, particular in cases where you need to stop a session immediately, a method has been added that allows you specify the timeout.

`System >> terminateSession:` *aSessionId* `timeout:` *aSeconds*

If the session does not respond to the request to terminate within the number of seconds specified in *aSeconds*, the sessions' gem process will be killed via a SIGTERM. *aSeconds* cannot be greater than the time specified in a STN_GEM_TIMEOUT, or 300 if the STN_GEM_TIMEOUT is not set. Note that STN_GEM_TIMEOUT is in minutes, so *aSeconds* <= STN_GEM_TIMEOUT * 60. If *aSeconds* is less than 0, the session is logged out, but the process is not sent kill -SIGTERM.

This method requires SystemControl privilege. It can be used to stop all sessions, including GcGem; but not the Symbol Gem.

To stop any session, including the Symbol Gem, the following new method may be used. This method may only be executed by SystemUser.

`System >> terminateSymbolCreationSession:` *aSessionId* `timeout:` *aSeconds*

New errors

The following errors have been added:

2423	RT_ERR_GCI_TRAV_NOT_LICENSED	License does not allow use of Gci Traversal operations.
4047	GS_ERR_SHRPC_LOSTOT_TIMEOUT	LostOt timeout detected when accessing the shared cache.

The following error now includes a String as the argument:

4037 GS_ERR_NO_CAPABILITY

Topaz changes

Topaz now allows multiple concurrent error handling statements. The following commands have been added or changed:

IFERR

The IFERR command has been added. This enables the use of 10 new post-error command buffers.

IFERR N <*response to error*>

The remainder of the command line following the integer N token is saved as an unparsed command line in the specified post-error buffer.

IFERR N

If integer N is the last token on the line clears the specified post-error buffer.

Whenever an error occurs (other than one matching an “expecterror” command and other than one during parsing of the IFERR command), or, whenever a result fails to match an “expectvalue” command, or, whenever a result matches an “expectbug” command, any non-empty post-error buffers are executed. Execution starts with buffer 1, and proceeds to buffer 10, executing each non-empty post-error buffer in order.

If an error occurs while executing one of post-error buffers, execution proceeds to the next non-empty post-error buffer. Error and result checking implied by DISPLAY RESULTCHECK, DISPLAY ERRORCHECK, EXPECTVALUE, etc., are not performed while executing from post-error buffers.

If a post-error buffer contains a command that would terminate the topaz process, then later buffers will have no effect. If a post-error buffer contains a command that would terminate the session, execution later buffers will be attempted but they will not have a session, unless one of the contains “login”.

IFERR_LIST

The topaz command IFERR_LIST has been added. This prints all of the non-empty post-error command buffers.

IFERR_CLEAR

The topaz command IFERR_CLEAR has been added. This clears all the post-error command buffers.

DISPLAY/OMIT ERRORCHECK

DISPLAY and OMIT have the added option ERRORCHECK. This is the same as RESULTCHECK, but no implied expectvalue true prior to the run or doit.

Changes to IFERROR

IFERROR <*response to error*> saves the <*response to error*> in post-error buffer 1. It is equivalent to IFERR 1 <*response to error*>. If <response to error> is blank, it clears post-error buffer 1.

Bugs Fixed

The following bugs in GemStone/S 64 Bit 2.2 have been fixed in GemStone/S 64 Bit 2.2.1.

Possible hang in Gem processes

It was possible for the stone to select a session for servicing and then lose the information about it, which resulted in the session waiting forever for a response from the stone. (#36720)

Unable to start stone on Linux without network

When a machine had no active network interface, and the result of `hostname` was not translatable to an IP address using /etc/hosts, stone startup would fail.

Changes in 2.2.1 will allow localhost to be used when a network is unavailable. There may be a 30 second or more delay before the first call times out and GemStone reverts to using localhost. (#36693)

Integer raisedTo: returned Float

This method always returned a Float. The ANSI specification states that for an Integer argument, Integer >> raisedTo: should return the same type as the receiver, an Integer. (#36660)

Locale not used in ScaledDecimal read/writes

ScaledDecimal read and write operations did not use the Locale decimalPoint setting. (#36666)

Some time zones encountered failures

With the new TimeZone code, some time zones with unusual daylight savings time offsets would encounter errors in creation or use. (#36712)

Poor performance in Fraction comparison

Fraction comparisons using < now use a faster algorithm, providing much better performance. (#36691)

Some complier errors reported as ObjErrBadOffset

Some compiler error descriptors were being returned with a size less than three. The missing third element of the error descriptor should be a string describing the error. (#36668)

RcQueue>>removeObject:addedBySessionId: incorrect behavior

This method did not ensure that the specified item was the item to remove, and did not update internal settings correctly. This could result in errors in subsequent RcQueue operations. (#36710)

Could not use slow stone executable in fast installation

It can be useful to use specific slow debugging executables within a regular fast product installation; this was not possible due to a mismatch in the size of an internal structure. (#36728)

Topaz died on login error

In some cases, login errors would cause topaz to exit, rather than wait for the problem to corrected and another login request made. (#36765)

Problems with very large OOPs

Several problem have been fixed involving OOP numbers larger than 4 billion:

- Lock removal incorrectly also removed other locks on the same object placed by other sessions. (#36780)
- DependecyLists (used in the indexing subsystem) could return object does not exist errors (#36753).
- Page audit failed following startstone and immediate stopstone (#36753).

Chapter 2

Object State Change Tracking

Introduction

GemStone transaction logs (tranlogs), which provide a way to recover all changes made to the repository in the case of repository crashes, or allow warm standbys to apply changes made to a primary repository, include detailed records of all changes in a encoded and compressed form. Converting the tranlog information to human-readable form, and analyzing this output, provides invaluable information for debugging and testing. It allows us to determine exactly what changes have been made to any of the objects in the repository over time. The tools used to perform this have been used internally to GemStone for many years, and have been provided to customers on occasion when needed to analyze specific application problems.

The ability to track changes to objects may be useful for customers who need to identify details on changes that have been made to application data objects. For this reason, we are making these scripts available as part of the GemStone/S 64 Bit product. Additional information has been added to the tranlogs to allow tracking of the specific user or machine that originated the object changes.

The scripts used to perform tranlog analysis are located in the `$GEMSTONE/bin/` directory and are named:

`printlogs.sh` — to output the complete contents (optionally filtered) of selected tranlogs in human-readable form.

`searchlogs.sh` — to search all tranlogs in a directory for selected OOPs (Object Oriented Pointers, or Object Ids), and output the matching entries (optionally filtered) in human readable form.

A GemStone repository performs many automatic operations, including things like garbage collection and checkpoints, that are transparent to end users. The tranlogs, of course, must contain records of any changes made by these operations. Complete details on everything that tranlogs may contain is beyond the scope of this document. The information provided here is intended to allow the use of the tranlog analysis scripts to locate and identify the details of changes to application objects.

Object oriented design's principle of encapsulation allows you to hide internal object complexity. However, to understand the data recorded in the tranlogs, you must have a detailed understanding of the actual structure of the objects. This includes both your own application classes, and the classes that are part of GemStone Smalltalk.

Also, note that since the tranlog analysis scripts are general purpose, used for a wide variety of purposes in which a detailed record of internal repository operation is required, the scripts may output much more information than is necessary for tracking object state changes. You may need to ignore this extraneous information as you perform your analysis.

Tranlog Analysis Scripts

Script Prerequisites

The environment variable `$GEMSTONE` must be set, and the `$GEMSTONE/bin/` directory must be in your executable path.

The scripts perform the analysis on tranlogs that are in the current working directory. If you are using raw partitions for your tranlogs, locate disks on the file system with adequate space for both the tranlogs themselves, and the script output files, which may be larger than the original tranlogs. Use `copydbf` to copy the tranlogs from the raw partition to the file system. For more information on the `copydbf` utility, see the *GemStone System Administration Guide.*

Output

Output from the scripts goes directly to `stdout`. To preserve the output and allow it to be used in later steps of analysis, redirect this; for example:

```
$> printlogs.sh tranlog1.dbf > tranlog1.out
```

Note that these scripts can produce very large amounts of output, so make sure that you have adequate disk space.

In some cases the resulting files may be too large for unix text editors such as `vi` or `emacs` to open. You may find it necessary to use the unix `split` utility to break up very large output files into more manageable chunks.

Tranlog Assumptions

By default, the tranlogs are assumed to be named using the GemStone convention `tranlog`*`NNN`*`.dbf`. If you are using a different naming convention, you can override this by setting the environment variable `$GS_TRANLOG_PREFIX` to the prefix you are using.

The scripts use the creation date of the tranlog file to determine the order in which the tranlogs are analyzed. If you copy or manipulate the tranlogs in a way that changes the creation date, this may cause the analysis to be done out of order. The output will warn of this with the message:

```
*** Warning:  scanning tranlogs out of order
```

Filter Criteria

The scripts both allow you to filter the results, to locate entries that are related to a particular UserId, GemHost, or ClientIP.

UserId – the userId (user name) of the UserProfile associated with this session: DataCurator, SystemUser, etc. The filter keyword is `user`.

GemHost – the name or IP address of the host machine running the gem process. For a linked session, which links the gem into the client, this is the same machine as the client.

If the gem is running on the same machine as the stone, use the name of the host machine. Otherwise, if the gem is on a different machine than the stone, use the IP address of the remote machine.

The filter keyword is `host`.

ClientIP – the IP address of the host machine running the client process.

For an RPC session, this is the machine running the client application. Clients may be topaz -r, GemBuilder for Smalltalk, or GemBuilder for C applications. For a linked session, this is the machine running the linked client / gem (the same machine as the GemHost). However, the ClientIP is always the IP address, even if it is on the same machine as the stone.

The filter keyword is `client`.

WARNING

Information about UserId, GemHost, and ClientIP are derived from a ***Login*** *tranlog entry created when a session first logs in. This entry associates the UserId/GemHost/ClientIP with a particular sessionID, which is then used as a key for subsequent tranlog entries. If you start analysis from a later tranlog which does not include this* ***Login*** *entry, these fields will be left blank for that session, and printouts/searches using filters based on these fields will not locate any results. Likewise, scanning through tranlogs out of order may result in the wrong* ***Login*** *entry being associated with a given sessionID. This would set UserId/GemHost/ ClientIP incorrectly for that particular session, and produce incorrect results when filtering.*

printlogs.sh

This script prints out the contents of one or more tranlogs in the current working directory in a human-readable form.

Warning

This script produce a very large amount of output, which (unfiltered) will exceed the size of original tranlog/s, and depending on the contents may be twice as large as the original tranlogs. Consider disk space, the use of filters, and restricting the set of tranlogs before running this script. Use caution in including the `full` *keyword.*

Usage:

```
printlogs.sh [<filters>] [full] [all] [<tlogA> ... <tlogZ>]
```

If **<*filters*>** are specified, only print out the tranlog entries that match the specified criteria. Filters may be combined. Possible filters are:

`user` <*userId*> - filter by the userId (the user name) of the GemStone UserProfile
`host` <*hostnameOrIP*> - filter by gem / topaz process host or IP address
`client` <*N.N.N.N*> - filter by client IP Address

`full` - adding this keyword produces more detailed information. WARNING: this produces much larger output results.

`all` - adding this keyword causes the script to print out the contents of all tranlogs in the current working directory.

Examples

To print out the entire contents of all tranlogs in this working directory:

```
printlogs.sh all
```

To print out all entries in a selected number of tranlogs (note that tranlogs in the sequence must be contiguous):

```
printlogs.sh tranlog5.dbf tranlog6.dbf tranlog7.dbf
```

To print out all tranlog entries for the user DataCurator in any tranlog:

```
printlogs.sh user DataCurator all
```

To print out detailed information for all entries in tranlog5.dbf for the user DataCurator:

```
printlogs.sh full user DataCurator tranlog5.dbf
```

searchlogs.sh

This script prints out tranlog entries associated with particular OOP values, according to the arguments in the command line. All tranlogs in the current working directory are scanned.

Usage:

```
searchlogs.sh [<filters>] <oopA> ... <oopB>]
```

If <*filters*> are specified, only print out the tranlog entries that match the specified criteria. Filters may be combined. Possible filters are:

`user` <*userId*> - filter by the userId (the user name) of the GemStone UserProfile
`host` <*hostnameOrIP*> - filter by gem / topaz process host or IP address
`client` <*N.N.N.N*> - filter by client IP Address

Examples

To print out all entries involving OOP 1234 and OOP 5678:

```
searchlogs.sh 1234 5678
```

To print out all entries involving OOP 1234 performed by DataCurator:

```
searchlogs.sh user DataCurator 1234
```

To print out all entries involving OOP 1234 and OOP 5678 performed by DataCurator while logged in from client machine 10.20.30.40:

```
searchlogs.sh user DataCurator client 10.20.30.40 1234 5678
```

Tranlog Structure

In order to make sense of the output from the scripts, you need a basic understanding of how tranlogs are structured.

GemStone transaction logs consist of a sequence of **tranlog records**. Tranlog records are are written on **physical pages** of 512 bytes (note that this is different from the larger page size used for extents); a tranlog record may extend over more than one page, but its size is always a multiple of 512 bytes.

Each tranlog record contains one or more **tranlog entries** (sometimes referred to internally as logical records). The tranlog entries contain the information of interest - the actual changes to objects in the repository (and any other repository operations).

Output from the scripts will include header information for the tranlog record (see Example 2.1), followed by data from each of the tranlog entries held by that tranlog record.

Example 2.1 Tranlog Record Header Output

```
Dump of page 3, Kind=(16)Tran Log Record
  thisRecordId:(file:2 rec:3) previousRecordId:(file:2 rec:2)
  recordSize: 512 numLogicalRecs: 1 fileCreationTime: 1156195399
```

Tranlog records are identified by the pageId that they begin on. Since a tranlog record may extend over multiple pages, there may be a gap in the sequence of pageIds in the output. For example, if the tranlog record starting on page 6 has a recordSize of 1024 (two 512-byte physical pages), then the pageId of the next tranlog record will be 8.

Example 2.2 Gap in Tranlog Record Sequence

```
Dump of page 6, Kind=(16)Tran Log Record
 thisRecordId:(file:3 rec:6) previousRecordId:(file:3 rec:5)
 recordSize: 1024 numLogicalRecs: 1 fileCreationTime: 1156738357


3.6.0  Login session: 5 user: DataCurator gemhost: myhost clientIP:
10.20.30.40 beginId:(26750 0)

---------------------------------------
Dump of page 8, Kind=(16)Tran Log Record
 thisRecordId:(file:3 rec:8) previousRecordId:(file:3 rec:6)
 recordSize: 512 numLogicalRecs: 1 fileCreationTime: 1156738357


3.8.0  Checkpoint session: 0 beginId:(26750 0)
  rootPageId: 3
  preChkLogRecordId: (file:3 rec:6) stoneStartup: 0
  logDebugLevel: 0  spare3..4: 0 0
```

Tranlog Entries

Each tranlog entry contains a unique identifying code, a descriptive entry type, and information on the session that originated the tranlog entry.

The identifying code consists of 3 numbers in the form:

```
AAA.BBB.CCC
```

where:

AAA – the fileId of the tranlog holding the entry
BBB – the pageId for the tranlog record holding the entry
CCC – the entryId: the number of the entry within the tranlog record

Each tranlog entry is of a particular type, according to the event that it represents and the information it contains. Types are indicated by short descriptive terms such as **Login**, **Abort**, **Commit**, and **Data**. There are a large number of entry types, most of which are not important for tracking object state changes and can be ignored (for example, the **Checkpoint** entry in Example 2.2). The ones that are important are documented below.

Each tranlog entry is associated with an Integer sessionID. SessionsIDs are unique to a specific session at any point in time. However, when a session logs out, the sessionID becomes available again for reuse by a new session logging in, so over a period of time, a sessionId may be used by a number of different sessions.

A sessionID of zero is used for tranlog entries generated by the stone - you may see this also in the **Checkpoint** entry in Example 2.2.

If the entry was not originated from the stone (that is, the tranlog entries sessionID is not 0), the tranlog entry header includes more information about the session: the UserId, the GemHost, and the ClientIP address. These are described in more detail under “Filter Criteria” on page 3.

Example 2.3 example tranlog entry

```
2.4.1  StartGcGem session: 1 user: GcUser gemhost: myhost clientIP:
10.20.30.40 beginId:(67 3)
```

This example shows that it is in the tranlog with fileId 2 (by the default naming convention, `tranlog2.dbf`), it is in the fourth physical page and in tranlog record 4, and it is the first tranlog entry in tranlog record 4.

This entry is of the tranlog entry type `StartGcGem`. The session is logged in as the user named `GcUser`; the gem session is executing on the same machine as the stone, a machine named `myhost`; and the client is executing on a machine with the IP address `10.20.30.40`. (beginId contains transaction tracking information that you can ignore).

Other information will follow this basic data, depending on the type of entry.

Tranlog Entry Types

There are a large number of tranlog entry types. Most of these are not relevant to tracking object state history - they record internal operations of the system, such as garbage collection or checkpoints. These tranlog entry types are not documented, although their functions can often be deduced by their names.

Below are the entries important for tracking object state history:

Login

A new session is logging into GemStone. As mentioned earlier, this entry sets the UserId, GemHost, and ClientIP data for this particular sessionID. If you start analysis on a tranlog that follows this event, these fields will be left blank for that session.

For example, if session 7 logs in while tranlog4 is in effect, and logs out when tranlog5 is in effect, and you begin analysis on tranlog5, entries for this session will not contain any UserId / GemHost / ClientIP information. If sessionID 7 is reused by a new session logging in later during tranlog5, that login will be recorded in tranlog5, and subsequent entries for this new session *will* be displayed properly.

There is no tranlog entry recorded for when a session logs out.

BeginData
Data
BeginStoreData
StoreData

GemStone uses the above four entry types for recording new or changed object data. The basic entry information is followed by additional information about the changes, including the OOP values of the changed objects. Using the optional “full” flag in the `printlogs.sh` script will cause the output to include additional information on the exact changes made.

Commit

All the changes recorded in earlier **BeginData**, **Data**, **BeginStoreData**, or **StoreData** entries are now officially committed to the repository.

Abort

Any changes recorded earlier in this transaction are discarded.

BreakSerialization

This entry indicates that a transaction conflict occurred during an attempt to commit. Any changes recorded earlier in this transaction are not yet permanent. This event is usually followed by an **Abort** entry, although it's possible that the next event seen for this sessionID is a **Login**, if the original session logged out and a new session reuses the sessionID.

Very Large Objects

GemStone is designed so that all objects will fit on a single page of 8192 bytes. This means that a byte object can be no larger than about 8000 bytes (since page header information uses some space), and pointer objects can only have about 2000 elements. GemStone internally represents objects larger than this as a tree structure, where the root node object references 2 or more leaf node objects, which then reference the actual elements of the collection object. Extremely large objects, such as large collections, may have internal branch nodes, if the number of leaf objects needed exceeds 2000.

This internal structure is usually transparent to the user. So, for example, you may create and manipulate an Array containing 10,000 elements as if it was a single large object,

while the actual representation is a root object that references five leaf objects, each containing a 2000-element chunk of the array. While this makes application development with GemStone much simpler, the entries in the tranlogs reflect the actual implementation; you will need to be aware of this to understand tranlog output relating to collections larger than ~2000 object references or ~8000 bytes. Adding an element to the large Array, for example, may mean the tranlog entry includes a change to an instance of LargeObjectNode (the leaf node object), rather than a change to the Array itself.

Full vs. Normal Mode

When using the printlogs.sh script, you can optionally specify "full" mode to get more details on the changes made to objects in the repository. But this will greatly increase the size of the resulting log files. For example, using normal mode on our test tranlogs generated a log file that contained an entry that looked like this:

Example 2.4 Tranlog entry, normal mode

```
5.7.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(151 2)

          clusterId: 1,  extentId: 0  numObjs:5
    154297  155069  155125  155185  155245
```

which tells you that 5 objects were created or modified during this event, with oops 154297, 155069, 155125, 155185, and 155245.

Using "full" mode will produce a much more detailed listing for this event:

Example 2.5 Tranlog entry, full mode

```
5.7.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(151 2)
          clusterId: 1,  extentId: 0  numObjs:5

objId: 154297 class: 1741 seg: 3253 sz: 14 bits: 0x27 psize: 76
Oop values:  19  7  11  23  10  154177  7845  155245
8: 10  10  10  10 78277  154233

objId: 155069 class: 182393 seg: 3253 sz: 5 bits: 0x67 psize: 40
Oop values:  155185  10  155125  156661  389295

objId: 155125 class: 1169 seg: 3253 sz: 8 bits: 0x3007 psize: 28
Bytes:  Oakville

objId: 155185 class: 1169 seg: 3253 sz: 16 bits: 0x3007 psize: 36
Bytes:  5234 Main Street

objId: 155245 class: 1745 seg: 3253 sz: 2 bits: 0x7 psize: 28
Oop values:  7845  155053
```

Note that there is now a description of each individual created or modified object, containing these fields:

`objId:` the OOP of this object

`class:` the OOP of the class of this object

`seg:` the OOP of the segment associated with this object

`sz:` the logical size of this object (in bytes for a byte object, OOPs for an OOP object)

`bits:` the format bits for this object (internal GS use)

`psize:` the physical size of this object on disk in bytes (including 20 bytes of object header information)

`Bytes:` the actual bytes that make up this object (if a byte object)
[or]
`Oop values:` the actual OOPs that make up this object (if an OOP object)

If the Oop values contain more than eight elements, they are broken into lines of eight items, each of which is prefixed by a counter. For example, an Array of 30 items might look like this:

```
objId: 210841 class: 1045 seg: 3261 sz: 50 bits: 0x47 psize: 220
Oop values:  10 10 210829  210825  210821  210817  210813  210809
8:  210805  210801  210797  210793  210789  210785  210781  210777
16:  210773  210769  210765  210761  210757  210753  210749  210745
24:  210741  210737  210733  210729  210725  210721
```

Bytes are broken up similarly, but the sections are 60 bytes rather than 8. for example, the source code string for the `name:` method may look like this:

```
objId: 141141 class: 1169 seg: 3261 sz: 91 bits: 0x3007 psize: 112
Bytes:  name: newValue

  "Modify the value of the instance variabl
61: e 'name'."
name := newValue
^@
```

Example of Tranlog Analysis

For this example, we created a simple database containing some Employee information and performed some simple operations creating and modifying these Employee objects.

The structure of classes associated with the Employee data is as follows:

Employee:
- name - a Name object
- age - a SmallInteger
- address - an Address object

Name:
- last - a String object
- first - a String object
- middle - a String object

Address:
- addr1 - a String object
- addr2 - a String object
- city - a String object
- state - a String object
- zip - a SmallInteger

After having User1 create 5 Employee objects (with associated Name and Address objects), we then had User1 and User2 make some minor changes to one of the Employees:

1. User2 incremented the age after a birthday

2. User1 changed the address after a move

When completed, we had four tranlogs, tranlog2.dbf to tranlog5.dbf. The following examples are drawn from these tranlogs.

Included in the `$GEMSTONE/examples/tranlogs` directory are two files containing the results from `printlogs.sh` on the example tranlogs. They can be found at:

`demolog.txt` – the results of: `printlogs.sh all`

`demologfull.txt` – the results of: `printlogs.sh full all`

Tracking Changes to an Employee

Let's say we want to examine the change history of a particular Employee. Using the method #asOop, we find the OOP of the Employee object of interest is 155053.

```
topaz 1> printit
| myEmployee |
myEmployee := <code to locate employee object>.
myEmployee asOop.
%
155053
```

The data composing an Employee is contained in subobjects (such as address), as well as directly (such as age). So, we will also need to track changes to these subobjects. Again

using #asOop, we find that the OOP of the Name object associated with this Employee is 155073, and that the OOP for the Address object is 155069.

```
myEmployee name asOop
155073

myEmployee address asOop
155069
```

You can use #asOop on any persistent object in the repository. For example,

```
73 asOop
295

nil asOop
20
```

We can now search our tranlogs for any events involving these objects. Using the command:

```
$> searchlogs.sh 155053 155073 155069
```

results in the following output:

Example 2.6 seachlogs.sh output for Employee

```
Searching for oops:  155053 155073 155069
```

```
Searching tranlogs:
  tranlog2.dbf
  tranlog3.dbf
  tranlog4.dbf
  tranlog5.dbf

3.61.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
  object 155053 cls 180689

3.61.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
  object 155069 cls 182393

3.61.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
  object 155073 cls 180101

3.61.1  Commit session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)

 crBacklog: 1  crPageId: 944
  beginLogRecord:(file:3 rec:61)  reclaimedOopsDebugInfo: 0
  timeWritten: [1156955988] Wed 30 Aug 2006 09:39:48 PDT

4.7.0  BeginData session: 2 user: User2 gemhost: merlin clientIP:
10.20.30.40 beginId:(142 2)
  object 155053 cls 180689

4.7.1  Commit session: 2 user: User2 gemhost: merlin clientIP:
10.20.30.40 beginId:(142 2)
 crBacklog: 1  crPageId: 872
  beginLogRecord:(file:4 rec:7)  reclaimedOopsDebugInfo: 0
  timeWritten: [1156956023] Wed 30 Aug 2006 09:40:23 PDT

5.7.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(151 2)
  object 155069 cls 182393

5.7.1  Commit session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(151 2)
 crBacklog: 1  crPageId: 875
  beginLogRecord:(file:5 rec:7)  reclaimedOopsDebugInfo: 0
  timeWritten: [1156956060] Wed 30 Aug 2006 09:41:00 PDT
```

From this output, we can see that in tranlog entry 3.61.0, User1 made changes to all three objects (in this case, when the Employee and associated subobjects were first created). In entry 4.7.0, User2 made a change to the Employee, and then later in entry 5.7.0, User1 made a change to 155069, the Address object.

Note that the **BeginData** entries are each followed by a **Commit**. You should always confirm that a **BeginData** / **Data** / **BeginStoreData** / **StoreData** of interest is followed by a **Commit**. If it doesn't then the reported event was not made persistent in the repository.

Changed vs. new objects

In the above example, while the field of an Address object changed, the Address object itself was the same (had the same OOP). Depending on how the Smalltalk application is written, this may not always be the case. If application that was initiating these changes created a new Address object, and assigned the Employee's address instance variable to this new object, then the Employee object would reference a new OOP, rather than OOP 155069. This would make the analysis somewhat different. For example, in the initial stage of the analysis when you look up the OOP of the Address object in your application, you would find the new OOP rather than the original OOP. Looking back in time, you would see when this Address object was created and assigned to the Employee instance.

Details of Changes to an Employee

Having used the `searchlogs.sh` script to get a general idea of which tranlogs are of interest, you can now use the `printlogs.sh` script to get more details.

Say you want more details on the creation of Employee object 155069 and its associated subobjects 155073 and 155069 in entry 3.61.0. The "3" in "3.61.0" indicates that `tranlog3.dbf` is the tranlog of interest. The command:

```
$> printlogs.sh tranlog3.dbf
```

will generate a condensed listing of all events in `tranlog3.dbf`. By searching the resulting file for the entry number 3.61.0 you can find the relevant entry:

Example 2.7 Employee modification

```
3.61.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
  clusterId: 1,  extentId: 0  numObjs:35

145021  155041  155045  155049  155053  155069  155073  155077  155089
155093  155097  155101  155121  155273  155285  155317  156589
156597  156609  156613  156661  156689  156729  156733  156749
156825  156829  156833  156837  156857  156861  156885  179629
180045  180653
```

This shows the oops of *all* objects created during this event. If you want to see more details on the actual changes made, use the "full" argument in the `printlogs.sh` command:

```
$> printlogs.sh full tranlog3.dbf
```

This will produce a more detailed listing of all events. For tranlog entry 3.61.0, you'll find:

Example 2.8 Example Employee modifications, output in full mode

```
3.61.0  BeginData session: 2 user: User1 gemhost: merlin clientIP:
10.20.30.40 beginId:(134 0)
     clusterId: 1,  extentId: 0  numObjs:35

        [details for other objects omitted]

objId: 155053 class: 180689 seg: 3253 sz: 3 bits: 0x47 psize: 32
Oop values:  155073  295  155069

objId: 155069 class: 182393 seg: 3253 sz: 5 bits: 0x47 psize: 40
Oop values:  156833  10  156837  156661  391335

objId: 155073 class: 180101 seg: 3253 sz: 3 bits: 0x47 psize: 32
Oop values:  156829  156825  10

        [details for other objects omitted]


objId: 156661 class: 1169 seg: 3253 sz: 2 bits: 0x3007 psize: 24
Bytes:  OR^@^@

        [details for other objects omitted]

objId: 156825 class: 1169 seg: 3253 sz: 7 bits: 0x3007 psize: 28
Bytes:  Patrick^@

objId: 156829 class: 1169 seg: 3253 sz: 5 bits: 0x3007 psize: 28
Bytes:  Ohara^@^@^@

objId: 156833 class: 1169 seg: 3253 sz: 13 bits: 0x3007 psize: 36
Bytes:  2556 Fir Blvd^@^@^@

objId: 156837 class: 1169 seg: 3253 sz: 7 bits: 0x3007 psize: 28
Bytes:  Ashford^@

        [details for other objects omitted]
```

So, for the Employee object 155053, we find:

```
objId: 155053 class: 180689 seg: 3253 sz: 3 bits: 0x47 psize: 32
Oop values:  155073  295  155069
```

Indicating that this is an instance of class 180689, the Employee class, which has three instance variables: name, age, and address. By position, we can identify the data in the instance variables:

name: 155073 - the OOP of an instance of Name, found later in the entry
age: 295 - the OOP of the SmallInteger 73
address: 155069 - the OOP of a instance of Address, found later in the entry

Looking at the Name object 155073 we find:

```
objId: 155073 class: 180101 seg: 3253 sz: 3 bits: 0x47 psize: 32
Oop values:  156829  156825  10
```

Indicating that this is an instance of the class with OOP 180101 (Name). Name contains three instance variables, last, first, and middle. By position, we see the data is:

last: 156829 - the OOP of a String, described below
first: 156825 - the OOP of a String, described below
middle: 10 (the OOP of nil) - in this example, no middle name was set

For the last name object 156829 we find:

```
objId: 156829 class: 1169 seg: 3253 sz: 5 bits: 0x3007 psize: 28
Bytes:  Ohara^@^@^@
```

Indicating the last name is the string “Ohara”. The ^@ indicate nulls in the String after its official end. There may be up to 3 of these nulls; Strings are adjusted to a size that is a multiple of four bytes.

By a similar process you can follow the trail of objects to examine the structure of other subobjects in the Name and Address objects.

Further Analysis

Class Operations

To find all objects created or modified that belong to a particular class, first generate `printlogs.sh` output in full mode of the tranlogs of interest. Each time an object of that class is created or modified, the full tranlog entry includes the line

```
class: <OOP>
```

Use the unix `grep` command to find all references to the OOP of the class.

For example, to find all creation or modification of any instance of our example class Employee:

First, generate `printlogs.sh` full mode output; in our case, `demologfull.txt`.

Since the Employee class is OOP 180689, execute the `grep` command:

```
$>  grep "class: 180689" demologfull.txt
```

this will produce output of the form:

```
objId: 155041 class: 180689 seg: 3253 sz: 3 bits: 0x47 psize: 32
objId: 155053 class: 180689 seg: 3253 sz: 3 bits: 0x47 psize: 32
objId: 155077 class: 180689 seg: 3253 sz: 3 bits: 0x47 psize: 32
objId: 155097 class: 180689 seg: 3253 sz: 3 bits: 0x47 psize: 32
objId: 155053 class: 180689 seg: 3253 sz: 3 bits: 0x47 psize: 32
```

This gives the first line from the entry creating / modifying the object belonging to that class. From this, you can use other commands to search for and / or track the history of these objects.

Deleted Objects

An object-oriented system doesn't actually delete objects; objects cease to be referenced and are eventually garbage collected. Noting the removal of an object requires examining the references to that object (such as from a collection) and identifying when the referencing object was modified in such a way that the object of interest is no longer referenced. Meanwhile, as you examine the `printlogs.sh` output, you may find references to the OOP of a dereferenced object in garbage collection tranlog entries.

Managing Volume

As noted above, the `printlogs.sh` produce a very large amount of output. GemStone tranlogs may be very large; while 2GB is historically recommended, up to 16 GB are supported. The output of `printlogs.sh` in normal mode will be somewhat larger than the original tranlog (the `printlogs.sh` output, being human readable, is less dense). The output from this script in full mode is much larger.

To manage the volume:

- avoid configuring your system with very large tranlogs.
- Ensure you have plenty of disk space available before beginning analysis.
- print only the tranlogs containing data you need. Use the `searchlogs.sh` script to identify exactly where the required information is located.

- Make sure only the relevant tranlogs are in the current directory; move the unneeded ones elsewhere. However, you must retain a continuous set of tranlogs without gaps in sequence, and you must include the tranlog with the original log entry, in order to have the UserId and other information provided.
- Once you have printed the output, use the unix utility `grep -n` to locate the lines of interest, and the unix utility `split -l` to break the resulting file up into more manageable size chunks.