GemStone/S 64 Bit™
Release Notes

Version 3.7.4

May 2025

INTELLECTUAL PROPERTY OWNERSHIP

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

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

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

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

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

COPYRIGHTS

PATENTS

GemStone software has been covered by U.S. Patent Number 6,256,637 "Transactional virtual machine architecture", Patent Number 6,360,219 "Object queues with concurrent updating", Patent Number 6,567,905 "Generational garbage collector with persistent object cache", and Patent Number 6,681,226 "Selective pessimistic locking for a concurrently updateable database".

TRADEMARKS

GemTalk, **GemStone**, **GemBuilder**, **GemConnect**, and the GemStone and GemTalk logos are trademarks or registered trademarks of GemTalk Systems LLC, or of VMware, Inc., previously of GemStone Systems, Inc.

UNIX is a registered trademark of The Open Group.

Intel is a registered trademarks of Intel Corporation.

Microsoft, **Windows**, **Windows Server**, and **Azure** are registered trademarks of Microsoft Corporation.

Linux is a registered trademark of Linus Torvalds and others.
Red Hat, **Red Hat Enterprise Linux**, **RHEL**, and **CentOS** are trademarks or registered trademarks of Red Hat, Inc.
Rocky Linux is a trademark or registered trademark of Rocky Enterprise Software Foundation.
Ubuntu is a registered trademark of Canonical Ltd., Inc.

AIX, **Power**, **POWER**, **Power8**, **Power9**, and **VisualAge** are trademarks or registered trademarks of International Business Machines Corporation.

Apple, **Mac**, m**acOS**, and **Macintosh** are trademarks of Apple Inc.

Instantiations is a registered trademarks of Instantiations, Inc.

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

Raspberry Pi is a trademark of the Raspberry Pi Foundation.
RabbitMQ is a trademark of VMware, Inc.
Prometheus is a registered trademark of The Linux Foundation.
Grafana is a registered trademark of Raintank, Inc. dba Grafana Labs.

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

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

Preface

About This Documentation

These release notes describe changes in the GemStone/S 64 Bit™ version 3.7.4 release. Read these release notes carefully before you begin installation, upgrade, or development with this release.

No separate Installation Guide is provided with this release. For instructions on installing GemStone/S 64 Bit version 3.7.4, or upgrading or converting from previous products or versions, see the Installation Guide for version 3.7.

Terminology Conventions

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

Technical Support

Support Website

gemtalksystems.com

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

- **Documentation** for the current and for previous released versions of all GemTalk products, in PDF form.
- **Product download** for the current and selected recent versions of GemTalk software.

- **Bugnotes**, identifying performance issues or error conditions that you may encounter when using a GemTalk product.
- **Supplemental Documentation** and **TechTips**, providing information and instructions that are not in the regular documentation.
- **Compatibility matrices**, listing supported platforms for GemTalk product versions.

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

Help Requests

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

Website: techsupport.gemtalksystems.com

Email: techsupport@gemtalksystems.com

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

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

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

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

24x7 Emergency Technical Support

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

Training and Consulting

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

Table of Contents

Chapter 1. Release Notes for 3.7.4

Chapter 2. Authentication using JSON Web Tokens (JWT)

Chapter

1 Release Notes for 3.7.4

Overview

GemStone/S 64 Bit™ 3.7.4 is a new version of the GemStone/S 64 Bit object server. Version 3.7.4 includes feature enhancements and bug fixes, including significant additional features for login.

These Release Notes include changes between the previous version of GemStone/S 64 Bit, v3.7.2, and v3.7.4. v3.7.3 was an alpha-only release; all changes in this version are included in these Release Notes.

If you are upgrading from a version prior to 3.7.2, review the release notes for each intermediate release to see the full set of changes.

The Installation Guide has not been updated for this release. For installation, upgrade and conversion instructions, use the Installation Guide for version 3.7.2.

Supported Platforms

Platforms for Version 3.7.4

GemStone/S 64 Bit version 3.7.4 is supported on the following platforms:

- Red Hat-compatible Linux 7.9, 8.10, and 9.5, and Ubuntu 22.04 and 24.04, on x86_64. GemStone is tested on a mixture of Red Hat, CentOS, and Rocky; these are all considered fully certified platforms. Any reference to Red Hat applies to any Red Hat-compatible distribution.
- Ubuntu 22.04 on ARM. Linux on ARM is for development only, not for production.
- macOS 15.3.2 (Sequoia) and 11.7.10 (Big Sur), on x86 and Apple silicon (ARM). macOS distributions are for development only, not for production.

Note that GemStone/S 64 Bit v3.7 and later on Linux are built using machine instructions that are not available on older x86_64 CPUs (more than about 12 years old); v3.7.4 will not run on these CPUs, regardless of the Linux OS version.

For more information and detailed requirements for each supported platforms, please refer to the *GemStone/S 64 Bit v3.7.2 Installation Guide* for that platform.

GemBuilder for Smalltalk (GBS) Versions

The following versions of GBS are supported with GemStone/S 64 Bit version 3.7.4:

GBS/VW version 8.8.1

VisualWorks 9.4.1 64-bit	VisualWorks 9.3.1 64-bit	VisualWorks 8.3.2 64-bit	VisualWorks 8.3.2 32-bit
▸ Windows 10, Windows 11 ▸ RedHat ES 7.9, 8.10, and 9.5; Ubuntu 22.04 and 24.04	▸ Windows 10, Windows 11 ▸ RedHat ES 7.9, 8.10, and 9.5; Ubuntu 22.04 and 24.04	▸ Windows 10, Windows 11 ▸ RedHat ES 7.9, 8.10, and 9.5; Ubuntu 22.04 and 24.04	▸ Windows 11

GBS/VA version 5.4.7

VAST Platform 11.0.1	VAST Platform 10.0.2	VA Smalltalk 8.6.3
▸ Windows Server 2016, Windows 10, and Windows 11	▸ Windows Server 2016, Windows 10, and Windows 11	▸ Windows Server 2016, Windows 10, and Windows 11

For more details on GBS and client Smalltalk platforms and requirements, see the *GemBuilder for Smalltalk Installation Guide* for that version of GBS.

VSD Version

The GemStone/S 64 Bit v3.7.4 distribution includes VSD version 5.6.1. This is the same version of VSD that was included with the previous release, v3.7.2.

VSD 5.6.1 is included with the GemStone distribution, and can also be downloaded as a separate product from https://gemtalksystems.com/vsd/

Updated library versions

The version of openSSL has been updated to 3.0.16

Added Shared Libraries

The distribution now includes additional shared libraries,

```
$GEMSTONE/lib/libcurl-3.7.4-64.so
$GEMSTONE/lib/libcurlutl-3.7.4-64.so
```

1. Additional Login Parameter Support

GemStone supports a number of types of authentication, in addition to the standard GemStone-native password: UNIX, LDAP, Single-sign on via Kerberos. This release adds support for login via JSON Web Tokens (JWT). These all utilize the same login parameters (userId and password).

In addition, you may also now provide passwords other than as strings. Passwords can be the value of an environment variable or in a text file on disk. The environment variable or filename is resolved using environment and filesystem access of the GCI client process, or the gem process that is creating the instances of GsTsExternalSession.

To use these alternate password sources, set the appropriate flag in the login parameters, and put the environment variable name or a filename in the password field. During login, the environment variable is accessed or the file with the given name is read and the result is used to login.

Ordinary logins using password strings do not require any changes from previous releases.

There are additional topaz commands, GsTsExternalSession parameters messages, or GCI login flags that allow you to specify the new password source options.

The environment variable $GS_DEBUG_LOGIN can be set in the client environment to print debugging information on the login to stdout of the Gem process. This is primarily intended to print the multiple validations that can be done by a JWT login; it also prints the authentication scheme that is used by the UserProfile.

Support for JWT Authentication

GemStone now supports authentication using JSON Web Tokens (JWTs), as another alternative authentication scheme for GemStone login. See Chapter 2 for details.

Note that this feature should be considered preview in this version of GemStone.While JWT logins have been carefully tested, the API may have unknown gaps and is subject to change.

The JsonWebToken class has been added to support testing for JWT login; this class also supports general uses of JWTs.

GsTsExternalSession added methods

The following methods have been added; these are convenience methods that invoke the method on the GsTsExternalSession's instance of GemStoneParameters.

`GsTsExternalSession >> jwtPassword:` *aString*
Consider *aString* to be a JSON Web Token (JWT) and set it as the password.

`GsTsExternalSession >> passwordEnvVar:` *aString*
Consider *aString* to be the name of an environment variable which contains the actual password. *aString* is evaluated in this current session's gem or topaz -l process.

`GsTsExternalSession >> passwordFileName:` *aString*
Consider *aString* to be the name of a file on the gem host which contains the actual password, which may be a JWT, GemStone, or other authentication password. The

filename will use the current directory of this session's gem or topaz -l process to evaluate *aString*.

`GsTsExternalSession >> setLoginDebug`
Enables writing debug info to the gem's log file.

GemStoneParameters added and changed methods

the GemStoneParameters have been extensively updated to support alternate password source locations as well as JWT authentication.

New methods accessing the new GCI login flags have been added, and some existing methods (`onetimePasswordFlag`, `onetimePasswordLoginFlags`, and `passwordlessLoginFlags`), have been renamed for consistency, and new `*flag` methods have been added. The `*flag` methods are not expected to be accessed directly.

The following instance methods have been added.

`GemStoneParameters >> passwordEnvVar:` *envVarName*
Configure the parameters to interpret the password instance variable as an environment variable, and use the value of *envVarName* in the password field. Applies to JWT and GemStone authentication schemes.

`GemStoneParameters >> passwordFileName:` *fileName*
Configure the parameters to interpret the password instance variable as a filename, and use the contents of *fileName* in the password field. Applies to JWT and GemStone authentication schemes.

`GemStoneParameters >> setPasswordIsEnvVar`
Configure the parameters to interpret the password instance variable as an environment variable and apply the value as the password field. Applies to JWT and GemStone authentication schemes.

`GemStoneParameters >> setPasswordIsFileName`
Configure the parameters to interpret the password instance variable as a filename, and apply the contents of that file as the password field. Applies to JWT and GemStone authentication schemes.

`GemStoneParameters >> setLoginDebug`
Enable writing login debug details to the gem's log file

`GemStoneParameters >> clearLoginDebug`
Disables writing debug info to the gem's log file.

The following methods support login via JWTs:

`GemStoneParameters >> setLoginWithJwt`
Configure the parameters to login with a JSON Web Token.

`GemStoneParameters >> jwtPassword:` *aString*
Configure the parameters to authenticate using JWT, and set the password instance variable to *aString*, which must be an encoded JWT.

Topaz added set subcommands

Topaz includes additional **set** subcommands to configure login.

SET ENVPASSWORD *onOrOff*

When ON, the password field is expected to contain the name of an environment variable that references the password. This password may be a native GemStone, UNIX, or LDAP password or encoded JWT. May be abbreviated as ENVP or ENVPASS.

SET FILEPASSWORD *onOrOff*

When ON, the password field is expected to contain the name of disk file that contains the password. This password may be a native GemStone, UNIX, or LDAP password or encoded JWT. May be abbreviated as FILEP or FILEPASS.

SET JWTJSONFILENAME *filename key*

Enables login using a JWT that is in a specifically structured JSON file. For more information on JWTs, see Chapter 2.

SET JWTPASSWORD *onOrOff*

When on, enables login using a JWT as the password. For more information on JWTs, see Chapter 2.

2. Other significant new features

Configurable remote commit

A configuration parameter has been added to control the location of the commit for Gems that are running on a different node than the Stone. The default for non-X509 Gems is false, so the critical section of a commit runs in the remote Gem, as in past releases. When GEM_REMOTE_COMMIT is set to true, the commit will run in Gem's page server (or HostAgent) on the Stone's node.

For Gems in X509-secured GemStone environments, the default is true, which is the behavior in previous releases. This can be configured so the commit runs in the remote X509 Gem by setting GEM_REMOTE_COMMIT to false.

GEM_REMOTE_COMMIT

If TRUE, a Gem on a remote cache will execute the critical region of commit in the session's thread in the gempgsvr or hostagent process on Stone host. Can only be enabled if gem and stone host have same byte order, and has no effect if the Gem is not remote from Stone

- For normal (non-X509) sessions, default is false.
- For X509 sessions, default is true if gem is remote.

Runtime equivalent: #GemRemoteCommit

Remote cache page server changes for keeping cache warm

Remote cache can be warmed on startup, but there was previously no mechanism to keep these warm as processes on the Stone's cache modify objects. This meant that a remote Gem server with cache warming was not effective over time in avoiding performance impacts of slow networks configurations.

Now, a page pushing mechanism has been added. When a remote Gem logs in and triggers the startup of a remote cache on that node, its configuration file may specify for the remote cache's SPC monitor to start page pusher and receiver threads, to keep the remote cache warm.

The default is 0, which provides the same behavior as in previous releases.

SHR_PAGE_RECEIVER_THREADS
: In a gem config file for a gem login that triggers startup of a remote cache, the number of page pusher and receiver thread pairs that will be started to keep the remote cache warm.

  Default: 0 max: 20, recommended: 2 to 4.

KeepAlive now configured per socket

Most firewalls terminate idle sockets after a period of time, which terminates a logged in Gem session. The system wide TCP_KEEPALIVE defaults to 2 hours, while most firewalls have a shorter idle socket timeout.

To allow this to be controlled within a GemStone environment rather than system-wide at the OS level, on Linux GemStone now sets SO_KEEPALIVE on individual interprocess sockets. The timeout default to 30 minutes, but can be configured by a GemStone configuration parameter, depending on the firewall timeout.

The following configuration parameter applies to all GemStone processes that open sockets.

SOCKET_KEEPALIVE

Timeout value in seconds to be used for setsockopt(fd, SOCKOPT_KEEPALIVE, timeout), for gem's socket to the GCI client, socket between gem and stone, socket between gem and a pgsvr or hostagent, socket between stone and a remote cache pgsvr.
SOCKET_KEEPALIVE may be used in stone, gem and x509 netldi config files.

Default: 1800 min: 15 max 2147483647

SIGHUP handling for log rotation

To support log rotation, on Linux the following GemStone processes now handle SIGHUP:

stoned, netldid, pgsvrmain, shrpcmonitor, gem, reclaimgem, admingem, symbolgem, hostagent, pgsvrmain

If the log file/s written to by this process no longer exist, a new file will be opened with the same name, using freopen().

The stone process's login log (enabled by STN_LOGIN_LOG_ENABLED) is reopened with a new timestamp in the file name.

3. Collection Changes

Collection printing changes

Collection printing changes

In previous releases, printing a collection using printString printed a limited string of around 700 characters, stopping after printing the collection element that brought the size over the limit. This produced highly variable result string sizes, depending on the size of the elements in the collection.

Now, printing stops in the middle of a collection element's printString, not at an element boundary, and the resulting string is reliably limited to the given size (700, or as specified by the `maxSize:` argument to `printingOn:maxSize:`, see bug below) plus 4 to allow for elipses and collection close character. The result string size will usually be smaller than in previous releases.

For example, given an array of integers,

previously: <*first part of printString*>, 1348, 1349, 1350, ...)

3.7.4: <*first part of printString*>, 1348, 1349, 13...)

Collection printing did not conform to PrintStream maxSize limit

Printing a dictionary or other collection using printString uses an instance of PrintStream and returns a string limited to about 700 characters, terminating in ...) to indicate truncation. This is designed to be controlled by creating an instance of PrintStream using `printingOn:maxSize:` and using that for printing; however, printing did not respect the `maxSize:` argument and still limited printing to about 700 characters. (#51281)

GsBitmap changes

Arguments to addAll: and removeAll:

Formerly, `GsBitmap >> addAll:` and `removeAll:` only accepted an Array argument. Now, these methods accept IdentitySet collections as arguments.

addAll: return value

`addAll:` also was documented to return the number of elements added to the GsBitmap, but it incorrectly returned the argument. This has been fixed. (#51339)

Added CharacterCollection and ByteArray HMAC methods

Additional HMAC methods have been added to ByteArray and CharacterCollection which support returning the HMAC as a ByteArray. Previous methods only allowed returning the HMAC as a LargeInteger or hexadecimal String. The new methods in ByteArray and CharacterCollection are:

```
asMd5HmacByteArrayWithKey:
asSha1HmacByteArrayWithKey:
asSha256HmacByteArrayWithKey:
asSha3_224HmacByteArrayWithKey:
```

```
asSha3_256HmacByteArrayWithKey:
asSha3_384HmacByteArrayWithKey:
asSha3_512HmacByteArrayWithKey:
asSha512HmacByteArrayWithKey:
```

Base64Url encoding

For JWT support, the following methods have been added:

`ByteArray >> asBase64UrlString`
Return a String which represents the receiver represented in base64Url format.

`CharacterCollection >> asBase64UrlString`
Return a String which represents the receiver represented in base64Url format.

4. Backup and Restore Changes

Improved performance for commitRestoreForFailoverAfterWaitingUpTo:

The method `Repository >> commitRestoreForFailoverAfterWaitingUpTo:` was added to make the failover process simpler, by combining the wait, stopLogReceiver and commitRestore operations. The performance was previously similar to the sum of the separate operations, and could be somewhat worse.

The performance is now much faster than the individual operations executed in series.

Added Repository methods

`Repository >> restorePosition`
Return the position of the tranlog being restored as a ScaledDecimal with scale 10.

`Repository >> restoreStatusDateAndTimeRestoredTo`
Returns a DateAndTime which represents time of the last checkpoint that the repository was restored to or nil if restore not active, or the restored-to time is not available. The result will be nil if restore is active but has not yet replayed any checkpoints from tranlogs.

`Repository >> restoreStatusPosixTimeRestoredTo`
Returns an Integer, or nil if restore is not active.

restoreStatusInfo additional field

The method Repository >> restoreStatus now includes an additional field:

```
17: a SmallInteger, posix time in seconds of time of last
   checkpoint restored to.
```

The existing field 11, which contains the same information in string format, is now legacy.

Added System methods

`System class >> lastCommitTranlogPosition`
Returns a ScaledDecimal representation of the Tranlog position of the commit record of this sessions' last successful commit. Zero is returned if this session has not committed since login, or if the stone process is configured to write tranlogs to /dev/null.

Methods that now accept DateAndTime in addition to DateTime

The following methods now accept a DateAndTime as well as a DateTime as an argument:

```
Repository >> restoreFromArchiveLogs:toPointInTime:
Repository >> restoreFromArchiveLogs:toPointInTime:withPrefix:
Repository >> restoreToPointInTime:
```

See also "restoreToPointInTime: failed with OS/Repository timezone mismatch" on page 27; changes in `_checkPointInTime:` affect this method.

5. Other Image Changes

See Chapter 2 for additional image changes related to JSON Web Tokens (JWT) and JWT login support.

Class categories

Client products such as Jade use Class categories as a way to navigate the class hierarchy, rather than SymbolDictionaries. Historically these were not maintained for most server classes, and class categories are not visible using GBS (other than when inspecting a class).

All kernel classes now have a preliminary category assigned. The category layout and assignments are not final; further modifications for consistency and usability should be expected.

Added GsTsExternalSessions methods

`GsTsExternalSession class >> newDefault:` *aGciTsLibrary* `forGemHost:` *aHostName*
Creates a new external session that is set to the user, and stone of the current gem with the password 'swordfish', and the argument library version. The gem host is set to *aHostName*. You may update any of these parameters before login for more complex environments.

`GsTsExternalSession class >> newDefaultForGemHost:` *aHostName*
Creates a new external session that is set to the user, and stone of the current gem with the password 'swordfish', and the same library version as the current image. The gem host is set to *aHostName*. You may update any of these parameters before login for more complex environments.

`GsTsExternalSession >> useOnetimePassword`
Obtains a onetime password for the UserProfile of the current session and sets it for the receiver. The UserProfile of the current session must have the #CreateOnetimePassword privilege else an exception is raised. The UserProfile of

the current session must also not be SystemUser, otherwise an exception will be raised at login time. The onetime password is valid for 300 seconds.

`GsNetworkResourceString class >> defaultGemNRSFromCurrentForHost:` *aHostName*
Return a new gem NRS with the gem's host being set to *aHostName*. Use default gem service, and default netldi, or as set by environment variable GEMSTONE_NRS_ALL.If GEMSTONE_NRS_ALL contains a #dir component, include that in result. Ignore any other parameters in the NRS of the current login.

ChildError now includes stdout

The instance variable has been added, and accessor methods:

```
ChildError >> stdout
ChildError >> stdout: aString
```

Additional field in descriptionOfSession:

The method System class descriptionOfSession: includes an additional field:

29. A SmallInteger, 0 or processId of the gempgsvr servicing this remote non-X509 session.

Kerberos Principal reporting methods

The following methods have been added:

```
KerberosPrincipal >> configurationReport
KerberosPrincipal >> printConfigurationOn: aStream
KerberosPrincipal >> printGroupsOn: aStream
```

FileSystem changes

Handling of Undefined environment variables

When an environment variable was undefined, this previously caused low level methods to return nil, which resulted in a message not understood.

Now, this will signal an error, `Error occurred (error 2710), Environment variable undefined in` *envName.*

Incorrect result for SystemResolver >> gemLogDirectory

Linked sessions do not have gem logs; this method was incorrect in linked sessions. It incorrectly returned the root directory, and could result in MNUs under some conditions. This method now returns nil for linked sessions. (#51363)

Removed origin methods

FileLocator and SystemResolver classes previously provided methods to access the paths for the GemStone extent and tranlogs. Since FileSystem is implement for the Gem's file system (not the server's), this was not correct and not reliable. (#51359)

The following methods have been removed:

```
SystemResolver >> extent1
SystemResolver >> extent1Directory
SystemResolver >> tranlog
SystemResolver >> _extent:
FileLocator class >> extent1
FileLocator class >> extent1Directory
FileLocator class >> extent:
FileLocator class >> tranlog
```

Changes in SystemResolver >> supportedOrigins

`SystemResolver >> supportedOrigins` returns different results in v3.7.4.

- `extent1`, `extent1Directory`, and `tranlog` are no longer included.
- `gemLogDirectory` is now included in supportedOrigins, only for RPC sessions.

Other added methods

The following methods have been added:

`Boolean >> asSymbol`
Returns #true if self==true, #false otherwise.

`GsFile class >> serverChangeDirectory:` *aString*
Changes the directory of the gem or topaz -l process on the server to aString. Returns true on success or false if the directory change was not successful. Raises an error is aString is not a kind of String or Utf8.

`GsTestClass class >> resultDir`
Return the directory into which results for tests will be written.

`Locale >> name`
Return the name of the Locale.

`SmallInteger >> asSymbol`
Returns a Symbol that indicates the numeric value of the receiver. Positive values do not include a leading +.

`System class >> epochGcEnabled`
Return true if epoch GC is enabled.

Removed Methods

The following methods have been removed:

```
FileLocator class >> extent1
FileLocator class >> extent1Directory
FileLocator class >> extent:
FileLocator class >> tranlog
GemStoneParameters >> onetimePasswordFlag
GemStoneParameters >> onetimePasswordLoginFlags
GemStoneParameters >> passwordlessLoginFlags
Repository >> _scaledDecimalFromFileId:blockId:
SystemResolver >> extent1
SystemResolver >> extent1Directory
SystemResolver >> tranlog
SystemResolver >> _extent:
TimeZone >> shouldWriteInstVar:
```

6. GcGem Improvements

AdminGem only runs when needed

Previously, the AdminGem was always running, although it was only needed to manage voting after an MFC, or when epoch is running. Now, the AdminGem is started up when needed, and shuts down when no longer needed.

ReclaimGem further performance and scaling improvements

In v3.7.2 and earlier 3.7.x versions, the ReclaimGem could reclaim pages at a rate that caused unreasonable extent growth. Many of these issues were addressed in v3.7.2; this release contains additional fixes to avoid issues of growth that were not fully resolved in v3.7.2.

ReclaimGem now limits frame demand when free frames are low

On a heavily loaded system, the cache may run low in free frames in the shared page cache. Since the system must search for free frames to clear, there is usually a serious performance impact. The ReclaimGem now monitors the FreeFrameCount stat produced by the shrpcmonitor and pauses when the free frame list is low. (#51272)

Reclaim of dead checkpoint frequency

The ReclaimGem reclaims dead objects after a mark/sweep, and shadow objects continually while the application is running.

Reclaim of dead objects now handles some limits differently than during reclaim of shadow objects. During dead reclaim, the free space threshold is set higher, checkpoints are triggered more often, and reclaim pauses to allow checkpoints to complete. (#51340)

STN_CR_BACKLOG_THRESHOLD was not working correctly

This configuration parameter sets a limit on the commit record backlog, above which commit records are disposed more aggressively. Since reclaimed pages are not available until after the commit record is disposed, this reduced reclaim efficiency. (#51285)

7. Utilities Changes

largememorypages -C now optional

The **largememorypages** script now uses a default of 1900 for the number of shared counters, and the **-C** argument is optional.

The computations done by this script also did not produce correct results for very large (terabyte-range) shared cache size arguments. (#51331)

startnetldi -l argument now accepts NRS %P and %M

When specifying the name of the NetLDI's log file using the **-l** argument, you may now include the %N and %P NRS identifiers in the argument filename. %P is expanded to the PID of the NetLDI, and %M is expanded to the host name that this NetLDI is running on.

Improvements to copydbf -i/-I output

Identifying extents in restore mode

When an extent is in restore mode, the status was previously reported as "Extent was not cleanly shutdown; recovery is needed", since the extent requires additional actions to be usable. Now, it more specifically reports:

```
Extent in restore from tranlogs, restored to tranlog position
   N.NNNN
Oldest tranlog needed to resume restore is fileId N (
   tranlogN.dbf ).
```

Checkpoint numeric position as well as timestamp

Previously, checkpoints were reported by start time; now, the numeric position of the checkpoint in the tranlog, extent or backup is included, as well as the timestamp. There are also minor changes in the timestamp printing format.

searchlogs output timestamps now include timezone information

The searchlogs script prints the date and time of operations (localized to the OS timezone); the output previously did not include timezone information. For example:

Old format: `2025-03-25-15:13:55.617`

New format: `2025-03-25T15:13:55.617-04:00`

8. GCI changes

Added login flags

Login flags have been to support password or JWT tokens for login in a file or environment variable.

Added login flags for GciLoginEx() and GsTsLogin()

The following login flags have been added. These flags apply to both classic GCI logins using GciLoginEx() and related methods, and thread-safe GCI logins using GsTsLogin() and related methods.

GCI_LOGIN_PW_FILE
Setting this bit indicates the password buffer contains a file name of a file which contains the actual password. The file name is evaluated against the file system or environment of the GCI client process. The type of password within the file is dependent on which other login flags are set. By default, the file is expected to contain a standard GemStone password string. If the GCI_JWT_PASSWORD flags is also set, then the file is expected to contain a JWT.

GCI_LOGIN_PW_ENVVAR
Setting this bit indicates the password buffer contains the name of an environment variable that references the actual password. The env var is evaluated against the file system or environment of the GCI client process. The type of password referenced by the env var is dependent on which other login flags are set. By default, the env var is expected to reference a standard GemStone password string. If the GCI_JWT_PASSWORD flags is also set, then the env var is expected to reference a JWT.

GCI_LOGIN_DEBUG
Setting this bit causes the gem process to write login debugging information to the GCI logger to assist with login debugging. No debugging information is written to the GCI client or its log file. Setting this bit also causes the gem to not automatically delete its log file.

The following only applies for JWT logins:

GCI_JWT_PASSWORD
Setting this bit indicates the provided password is a JWT in base64url format. The JWT may be provided in a buffer, in a file, or referenced by an environment variable depending if any of the other new login bit flags have been set.

GCI login flags are now validated to ensure only legal combinations of flags can be used. Invalid combinations will result in an error.

Added GCI thread-safe functions

These functions are identical to the similar existing GCI TS calls (GciTsLogin() and GciTsNbLogin()) except that the new calls accept netldiName as an additional argument.

```
(GciSession) GciTsLogin_(
   const char *StoneNameNrs,
   const char *HostUserId,
   const char *HostPassword,
   BoolType hostPwIsEncrypted,
   const char *GemServiceNrs,
   const char *gemstoneUsername,
   const char *gemstonePassword,
   const char *netldiName,
   unsigned int loginFlags /* per GCI_LOGIN* in gci.ht */ ,
   int haltOnErrNum,
   BoolType *executedSessionInit, /*output*/
   GciErrSType *err);

(GciSession) GciTsNbLogin_(
   const char *StoneNameNrs,
   const char *HostUserId,
   const char *HostPassword,
   BoolType hostPwIsEncrypted,
   const char *GemServiceNrs,
   const char *gemstoneUsername,
   const char *gemstonePassword,
   const char *netldiName,
   unsigned int loginFlags /* per GCI_LOGIN* in gci.ht */ ,
   int haltOnErrNum,
   int *loginPollSocket /* output */);
```

9. Cache Statistics Changes

Cache Statistics Changes and Fixes

Stone cache statistic TimeInCheckpoint not working

This statistic was not incremented for most checkpoints. (#51342)

Added cache statistics

The following cache statistics have been added:

CommitsAfterReplay (Gem)
The number of commits that succeeded after executing RC replay.

CompletedCheckpointCount (Stn)
The number of checkpoints that have been completed since the stone was last started. See CheckpointCount

DataPagesCommitted (Gem)
Total number of data pages made persistent (committed) by the session. This statistic is updated during commit processing.

TimeInLastCheckpointMs (Stn)
Time in milliseconds from start to end of most recent checkpoint.

Removed cache statistics

The following cache statistics have been removed:

LastCommandFromClient

PageDisposesDeferred

10. Bugs Fixed

The following bugs were present in v3.7.2 and are fixed in v3.7.4:

Gem runs hot in detached execution

When a session logs in using the recently added GCI login flag GCI_PERFORM_DETACH, the Gem may run hot and consume a large percent of CPU due to excessive socket polls. (#51287)

Multithreaded operations on remote Gem with encrypted extents corrupts cache

When a Gem running on a host other than the Stone's host, and the Stone is using encrypted extents, executing a multithreaded scan operation (such as listInstances, allReferences, or markForCollection) in the remote Gem may cause cache corruption. (#51417)

ReclaimGem was not responsive while in sleepTimeBetweenReclaimUs

To control the impact of reclaim, the ReclaimGem can be configured to sleep between reclaim operations, using the parameter #sleepTimeBetweenReclaimUs. This has a maximum setting equivalent to 5 minutes. There was a bug such that the ReclaimGem did not respond to sigAborts while sleeping. If #sleepTimeBetweenReclaimUs is set to a value greater than the sigAbort timeout, on a busy system, the ReclaimGem could be killed repeatedly by lostOT processing. (#51328)

GEM_TEMPOBJ_POMGEN_SIZE always about 83%

When GEM_TEMPOBJ_POMGEN_SIZE is set to the default (0), the value should be calculated as a percentage of the GEM_TEMPOBJ_CACHE_SIZE, with the percentage decreasing as the cache size increases. However, this decreasing percentage was not applied, and the value of GEM_TEMPOBJ_POMGEN_SIZE was always about 83%. (#51375)

ReclaimGem fails to release commit token on fatal error in thread

If a ReclaimGem thread gets a fatal error (this has been seen only in a case of cache corruption), the ReclaimGem does not correctly release the commit token, preventing further commits until the ReclaimGem is manually killed or the Stone is restarted. (#51422).

Gem may be unresponsive to SIGTERM when executing in a primitive

When a Gem was executing in a primitive, it could become unresponsive to SIGTERM, and require kill -9. Now, 5 seconds after a second SIGTERM the process will exit. (#51341)

installgs reported that a valid keyfile was invalid

The installgs script was affected by changes in v3.7.2 for the new configuregs script, and the validation check when installing a keyfile reported that the keyfile was invalid. The keyfile is installed and provided it is valid, will work as usual. (#51355)

Timezone and DateTime related issues

restoreToPointInTime: failed with OS/Repository timezone mismatch

The method `Repository >> restoreToPointInTime:` does argument validation checks to ensure the argument is with the range of the repository birth time to the last time restored to in the repository. The check for the restore time did not calculate the time correctly when the repository was not configured to be in the same timezone as the OS. The validation was the only thing that failed, the actual restore operations did not have an issue. (#51350).

This method also now accepts an instance of DateAndTime, described on page 17.

DateTime/TimeZone passivate-activate issues

When a DateTime is passivated, the associated instance of TimeZone, which is the current Timezone when the DateTime was created, is also passivated. The Timezone's passivation was incomplete; recreatable data was not explicitly included. While the reactivated DateTime worked correctly, some messages sent to its TimeZone failed. (#51389)

Time instances with non-Integer milliseconds could break DateTime creation

In versions earlier than 3.7, it was possible create instance of Time with a non-Integer milliseconds field, such as a SmallDouble. After upgrade to 3.7.2, using these instances to create an instance of DateTime errored. (#51332)

System class >> processKindForSlot: could return zero

There was a very small window during the period where a session is logging in, for which `System class >> processKindForSlot:` for the new session's slot (invoked by another session) may return 0. The method `System class >> cacheStatisticsDescriptionAt:` expects a non-zero result, and reported a primitive error. (#51324)

DBF_EXTENT_SIZES was not cleared correctly on Stone restart

When DBF_EXTENT_SIZES is set and the repository is pregrown to that size, removing that configuration from the configuration file and restarting the stone reported that it was setting to unlimited, but in fact it restricted growth to the previously set limit. (#51343)

GsFileIn did not handle server file access errors in RPC environments

The class GsFileIn provides the ability to file topaz-format code into GemStone programmatically, from files on the client or server file systems. It uses GsFile to open and read files. If an error occurred when opening a server file, the incorrect (client) method was used to fetch the error; this method worked in linked sessions, but returned nil in RPC sessions. In a thread-safe environment, since client GsFile operations are disallowed, this error returned was a less understandable GciTransportError. (#51345).

pthread_join or other issues due to thread not shutdown

For a remote session that is connected to a mid-level cache on a third node, there is a thread for that session's slot in the Stone's cache that services reads from the mid-level cache. This thread was not being shut down at session logout. This could result in pthread_join issues or other issues. (#51367)

Upgrade could fail if Published SymbolDictionary is missing

If the Published SymbolDictionary was removed, upgrade could have failed.

Remote cache warming did not report progress count correctly

The log files for cache warming on remote gems incorrectly reported progress count as 0 until complete. (#51306)

Sending _selectiveAbort to a temporary object SEGVed

`Object >> _selectiveAbort` is not intended to be used on temporary objects; this method SEGVed on the attempt. #51274)

Split-tranlogs logsender slow to see commits

In a split tranlogs configuration, a logsender writes tranlogs in multiple files based on a time interval, which is separate from a logsender's role in a hot standby configuration. The split tranlogs logsender could be slow to detect Stone commits, and commits could be missing in the split tranlogs after Stone shutdown. (#51388)

numElements and numCollisions may have become incorrect in a StringKeyValueDictionary

It was possible for the numElements and numCollections instance variables to become incorrect in an instance of StringKeyValueDictionary under certain conditions of removing entries. (#51278)

In-memory GC may have failed during load of a large method

If garbage collection of temporary object memory runs while a large method was being loaded, there is a small window in which it may have encountered an uninitialized value and fail. (#51270)

Improved error message for error #2110, OBJ_ERR_BAD_OFFSET

This error was signalled with incorrectly ordered arguments, and the resulting message was unclear. (#51409)

X509-Secured GemStone Issues

X509-Secure GemStone remote commits could error

In X509-Secured GemStone, Gems on nodes remote from the Stone execute the commit in the hostagent by default. This could result in connection errors during a Gem commit, or thread shutdown of page pusher/receiver threads for remote cache warming. (#51353)

GsTsX509ExternalSession spurious "call already in progress" errors

The initialization of a GsTsX509ExternalSession is incomplete, which resulted in invalid "call already in progress" errors. (#51382)

Killing a remote cache pageserver could cause stone issues

When a remote pageserver that is part of an X509-secured GemStone configuration is killed, it could cause Stone issues; the Stone may not be aware the remote cache is gone, and sessions may be stuck in after logout. (#51385)

Port range for mid-level cache warmer gems

When cache warming an X509-secured mid-level cache, the Gem on the mid-level cache node that is handling the warming listens for connections to the cache warmer pusher threads on the Stone's node. This now uses the port range specified by NETLDI_PORT_RANGE.

Mid-level cache warming threads could have exited prematurely

It was possible for the threads warming a mid-level cache to exit before transferring all of the data pages found in the Stone cache. (#51364)

TotalSessionCount not decremented on when cache or hostagent killed

When an X509 remote cache or hostagent is killed, the Stone's TotalSessionCount is not decremented for the sessions that were logged in using that hostagent. This can lead to Too many sessions logged in errors. (#51387)

Risk of hostagent SEGV when killed

When the hostegents gets a kill -TERM immediately after starting, there is a risk it may SEGV rather than terminating normally. (#51395, #51402).

X509 Netldi may have shutdown when midcache hostagent startup fails

When an X509 mid-level cache hostagent fails during startup, the X509 NetLDI may also shut down. (#51400)

Chapter

2 Authentication using JSON Web Tokens (JWT)

GemStone/S 64 Bit vers 3.7.4 adds the ability to login using JSON Web Tokens (JWTs).

This feature should be considered preview in this release. While JWT logins have been carefully tested, the API is subject to change.

A JSON Web Token (JWT) is encoded JSON that contains a set of required and optional key/value pairs, and is digitally signed using a key. GemStone authentication supports signing with an RSA public/private key pair.

The JWT's JSON contents has three sections: the header, the payload, and the signature. The JWT is transmitted in Base64Url encoded format, with each section separately encoded and period-delimited as *EncodedHeaderSection.encodedPayLoad.encodedSignature*. This encoded string is what is meant by JWT in this document.

Authentication in GemStone using JWTs

JWTs can be used for authentication in GemStone the way a GemStone password is used, for UserProfiles that are configured with this authentication scheme. Since some of the authentication must be done before the UserProfile is accessed, there is an additional flag that must be set for login using JWT.

JWT authentication is supported when logging in from topaz, GBS v8.9alpah1 and later, GsTsExternalSession, and using the classic and thread-safe GCIs.

JWT authentication cannot be used for system UserProfiles (SystemUser, DataCurator, GcUser, SymbolUser, Nameless, and HostAgentUser).

In addition to using the JWT as a password, GemStone login now allows alternative password source locations; you may set an environment variable to reference the encoded JWT, or the encoded JWT can be put in a disk file, or the encoded JWT can be embedded as the value in a JSON key-value pair in a disk file. For more details on the new password source location support, see "Additional Login Parameter Support" on page 11.

To use JWTs for GemStone authentication, you must do the following:

1. Acquire a JWT

Generally, you will acquire the JWT from a source outside GemStone. This JWT service also provides the public key corresponding to the private key used to sign the JWT, via

OpenId discovery documents. The details of acquiring a JWT from an OpenId service provider is outside the scope of this document. See "JWT Tokens" on page 33 for minimum expected information.

Using the JsonWebToken class, you can create a JWT and sign it using your own private key, and configure the Stone with the corresponding public key. This allows the hand-constructed JWT to be used for authentication. This mechanism is designed for testing; keys added to the Stone in this way are temporary; they do not persist across Stone restart. These temporary JWTs are not authenticated outside of GemStone and do not require OpenId discovery configuration.

2. Configure the Stone

For JWT authentication using OpenId document discovery to acquire public keys, the Stone must be configured for JWT public key discovery. See "Configuring the Stone to support JWTs" on page 34.

No Stone configuration is required for authentication using public keys manually added to support temporary JWTs.

3. Configure the UserProfile with a JwtSecurityData

The UserProfile must be configured specifically for the JWT requirements of the JWTs that will be used for login.

You must create a JwtSecurityData object and configure it with the accepted values for each claim that will be defined in the JWT. This JwtSecurityData is added to the GemStone UserProfile when configuring it to authenticate via JWT.

See"Configure the UserProfile with a JwtSecurityData" on page 37.

4. Configure login parameters

Unlike other types of authentication for which the UserProfile's configured authentication is recognized, JWT authentication must be explicitly enabled in the client before login.

Note that there are additional password sources supported in v3.7.4, including passwords in filenames or environment variables; which may optionally also be configured.These are described under "Additional Login Parameter Support" on page 11.

The specifics of logging in depends on the client:

- To login using a JWT from **topaz**, use added topaz commands. See "Login via Topaz" on page 43.
- To login using a JWT from an **external session**, use added methods in GsTsExternalSession and GemStoneParameters. Login using GsExternalSession is not supported for JWT logins. See "Login via GsTsExternalSession" on page 45.
- To login using a JWT in a **GCI application**, there are additional login flags. See "Login via GemBuilder for C" on page 47.
- **X509-secured** logins are not currently supported for JWT authentication.

JWT Tokens

The JWT header and payload sections contains required claims, and may also contain an arbitrary number of optional claims. The following claims are required

Table 1 Required JWT claims

Claim Key	Section	Required Status	
"kid"	header	required, automatically managed when using OID public keys.	String with key ID
"alg"	header	required	"RS256"
"typ"	header	required	"JWT"
"exp"	payload	required; automatically enforced.	Number defining the expiration time
"iss"	payload	required, and claim must be configured in JwtSecurityData.	String with issuer
"aud"	payload	required, and claim must be configured in JwtSecurityData.	String with audience

The JSON, for example would be something like this:

```
{
"header":
   {
   "kid":"theId",
   "alg":"RS256",
   "typ":"JWT"
   },
"payload":
   {
   "exp":expiration,
   "iss":"URLOfIssuer",
   "aud":"intendedAudience",
   },
"signature": aByteArray
}
```

Additional payload claims

The JWT will also normally contain additional payload claims, which depend on the service generating the JWT.

If an optional claim does not have a corresponding claim in the JwtSecurityData used for validating login, then the JWT claim is ignored for login.

The exception is "nbf", Not Before Time. "nbf" is not required, but if the JWT includes "nbf", the value is automatically enforced during login; this does not require configuration on the JwtSecurityData.

Ordering of JWT claims

The order of the fields in JSON and a JWT is not defined or intended to be meaningful. By convention, some fields are ordered before others, but this is not guaranteed. This means that the encoded JWT, and any JSON printed from the JWT, may not be identical. This does not affect the utility of the JWT; the login process relies on the existence of the keys and not on the specific layout within the JSON.

More information

For more on JWTs, including the commonly used additional claim keys, see:

https://auth0.com/docs/secure/tokens/json-web-tokens

https://www.rfc-editor.org/rfc/rfc7519.txt

2.1 Configuring the Stone to support JWTs

To allow login using a JWT that was signed with a private key, the Stone needs to have access to the corresponding public key.

The public key discovery mechanism is handled by a new thread in the stone, the OpenId Discovery (OID) thread. This thread handles discovery of the OpenID documents, and updates the list of public keys stored in the Stone to add new keys and remove old keys.

Temporary keys that are manually added to the Stone are handled separately and are not affected by the OID thread.

JWTs authenticated via OpenId discovery require that the Stone must be configured with one or more STN_OPENID_DISCOVERY_URLS; without this, the OID thread does not start.

Public Key Discovery

Providers of JWTs share the public keys through a key discovery URL, from which can be found the URLs of JSON Web Key Set (JWKS) documents, which contain one or more public keys.

JWT providers periodically rotate keys. Most JWKS documents which provide public keys for verifying JWTs contain two public keys at any one time: the public key currently in use and the next public key to be used sometime in the future. The period of validity for these public keys is specified by the key provider. A typical duration is 30 days, however some providers rotate keys as frequently as once per hour or as seldom as once per year.

The Stone maintains a list of public keys corresponding to the key Ids that are issued in JWTs. The public keys are read by the OID thread, which is responsible for making the TLS handshake to the key discovery URL and parsing the resulting JSON. Since the public keys are rotated, the table of public keys must be periodically refreshed.

OpenId Discovery Thread

The OID thread is responsible for making the connection to the discovery document server, parsing the resulting file, keeping track of currently valid public keys, and periodically refreshing these keys.

When the Stone is configured with one or more OpenId discovery URLs (STN_OPENID_DISCOVERY_URLS**)**, the OpenId Discovery (OID) thread is started. This thread writes to its own log file, *stoneName_stonePID*`openidthd.log`. If STN_OPENID_DISCOVERY_URLS is not defined, the OID thread does not start.

At Stone startup time, the OID thread queries all configured URLs in STN_OPENID_DISCOVERY_URLS to obtain the OpenId discovery documents. The specified URLs may reference either a JSON key discovery document or a JSON JWKS document. No other document formats are accepted.

These documents are parsed to collect the elements of the public keys, which are stored in the Stone. These public keys are used to authenticate with the private key in the JWT signing certificate.

If any of the URLs cannot be read or the documents found cannot be parsed, Stone startup will fail. After the Stone has started, failure to access and parse the documents is not a fatal error; a warning is printed to the OID thread log. JWT login attempts will use the previously loaded public keys.

Configuring Stone

Configuring the discovery URL or URLs is done with the Stone's configuration parameter STN_OPENID_DISCOVERY_URLS. This must be done before Stone starts.

For example, if using google's openID, add the following line to system.conf:

```
STN_OPENID_DISCOVERY_URLS = "https://accounts.google.com/.well-known/openid-configuration";
```

The configuration parameter STN_OPENID_DISCOVERY_INTERVAL sets the frequency that the discovery documents at the given URL or URLs is refreshed. The default of once per hour is used, if this is not explicitly defined. See "STN_OPENID_DISCOVERY_INTERVAL" on page 37

Accessing the OpenId discovery document

Although no authentication is required to access the OpenId discovery document, most sites host this document on a web server which allows only secure (https) connections. In order for the OID thread to connect with hosts in STN_OPENID_DISCOVERY_URLS, a TLS handshake is required. This handshake requires access to the trusted root certificate bundle file on the Stone's host.

The Stone startup will fail if the root certificate bundle file cannot be found or does not enable the handshake to access the OpenId discovery document.

By default, the Stone will look in the default location per the OS, so on most systems, nothing further is required. If you have a different setup, you must set the correct path; see "STN_OPENID_DISCOVERY_CA_CERT_FILE" on page 36.

You may also specify the location by setting the environment variable GS_CURL_CA_FILE to reference the location of the certificate bundle file. This must be set before the Stone starts up. If this environment variable is defined and the GS_CURL_CA_FILE destination is invalid, Stone startup will fail.

Special case on Darwin

On Darwin, the required certificate bundle is not installed by default. MacOS has its own proprietary framework for managing X.509 certificates (key chain), which is not used by OpenSSL. Darwin users must install the open source homebrew package "ca-certificates" in order to acquire a valid certificate bundle. Note that many open source Darwin packages also require this package, so it may already be installed on development systems.

JWT related queries on the Stone

The following methods provide support for repository-wide JWT support in the Stone.

`System class >> jwtKeyRefreshEnabled`
: Return a boolean indicating if the Open ID discovery thread in stone is enabled for this repository. If this returns false, only manually-added temporary JWT public keys will allow login via JWT.

`System class >> forceOpenIdKeyRefresh`
: Forces the Open ID refresh thread in stone to refresh the list of public keys used to authenticate JSON Web Token (JWT) logins. Returns true on success or false if discovery-based JWT logins are not enabled.

`System class >> jwtLoginsEnabled`
: Return a boolean indicating if logins with JSON Web Tokens (JWTs) via OpenId discovery are enabled for this repository.

`System class >> jwtPublicKeys`
: Returns the keys and key ids maintained by stone to validate JWT logins.
There are two lists of keys in stone:
1 - keys obtained by the Open ID Discovery thread.
2 - temporary keys manually added to the Stone.
The result of this method includes keys from both lists. The result is an Array of pairs where the odd numbered elements are key IDs (Strings) and the even numbered elements are instances of GsTlsPublicKey.

Added Configuration Parameters

The following configuration parameters have been added to support JWT logins.

The STN_OPENID_DISCOVERY_URLS parameter is required to enabled OpenId discovery based for JSON Web Tokens (JWTs) for login authentication; other parameters have defaults. If the parameters are defined, the values must be configured to support JWT authentication. These parameters are not required to use temporary testing JWTs.

STN_OPENID_DISCOVERY_CA_CERT_FILE

Specifies the location of the CA certificates file to be used when sending requests to obtain public keys to the URLS specified in the STN_OPENID_DISCOVERY_URLS configuration parameter.

If no option is specified (the default), the stone looks in the default locations:

Linux (Fedora: Red Hat, etc):

```
/etc/pki/tls/certs/ca-bundle.crt
```

Linux (Debian: Ubuntu, etc):

```
/etc/ssl/certs/ca-certificates.crt
```

MacOS (Darwin):

```
/opt/homebrew/etc/ca-certificates/cert.pem
$HOMEBREW_PREFIX/etc/ca-certificates/cert.pem
```

While this configuration parameter is optional. if it is set, it must be set to an existing file; otherwise Stone startup will fail.

STN_OPENID_DISCOVERY_INTERVAL

Specifies the frequency in minutes that the OpenId discovery thread in stone will refresh the discovery documents referenced by STN_OPENID_DISCOVERY_URLS. Discovery documents are always refreshed at Stone startup time. This parameter has no effect unless STN_OPENID_DISCOVERY_URLS is specified.

Runtime equivalent: #StnOpenIdDiscoveryInterval
(requires SystemControl privilege)
Default: 60
Min: 1
Max: 1000000
Units: Minutes

STN_OPENID_DISCOVERY_URLS

Specifies a list of URLs used to obtain OpenID discovery documents (OIDD) or JSON Web Key Sets (JWKS). Each URL specified must reference either a OIDD or JWKS. OIDDs contain the URL address of a JWKS and are preferred.

URL arguments must be in this format:

```
"https://<url>"
```

One or more entries are required in order to use OpenJSON Web Tokens (JWTs) as login credentials. Discovery documents are used to obtain public keys that are required in order to verify the signatures of JWTs.

Default: None

2.2 Configure the UserProfile with a JwtSecurityData

To enable JWT authentication for a UserProfile, you must create and configure an instance of the new class JwtSecurityData, and add this to the UserProfile. This JwtSecurityData must have the acceptable values defined for all the claims in the JWT that will be used for login.

JwtSecurityData

The class JwtSecurityData encapsulates the specific security requirements to allow login with a JWT. The actual claims in a JWT will depend on the JWT OpenId service provider that provides the JWT, and the details used to request the JWT from the OpenId service.

Each claim in the JWT must have a corresponding key in the JwtSecurityData, and the JWT's value at that key must match the JwtSecurityData's allowed values.

The JWT's expiration, and not before time (nbt) if present, are automatically checked and do not need to be configured in the JwtSecurityData.

A JwtSecurityData is configured with:

- **One or more issuers**
  `validIssuers` is an IdentitySet of Symbols containing acceptable values for the 'iss' JWT member. If the wildcard #* is included, any value will be accepted.

  To add an issuer, use `addIssuer:` *stringOrSymbol*. You may call this more than once to allow multiple issuers.

- **One or more audiences**
  `validAudiences` is an IdentitySet of Symbols containing acceptable values for the 'aud' JWT member. If the wildcard #* is included, any value will be accepted.

  To add an audience, use `addAudience:` *stringOrSymbol*. You may call this more than once to allow multiple audiences.

- **One or more userIds**
  `validUserIds` is an IdentitySet of Symbols containing acceptable values for the userId, which is the JWT member at the receiver's setting for `userIdKey`. The userIdKey is 'aud' by default. If the wildcard #* is included, any value will be accepted. This does not need to match the UserId of the UserProfile to which this JwtSecurityData will be used.

  To add an userId, use `addUserId:` *stringOrSymbol*. You may call this more than once to allow multiple userIds.

- **Zero or more user claims**
  `userClaims` is an Array of instances of JwtUserClaim Objects. Each claim specifies the corresponding JWT key and the accepted values. User claims are optional but if present in the JwtSecurityData, the JWT will be validated for that claim.

  To add a claim, use `addUserClaim:` *aJwtUserClaim*. You may call this more than once to allow multiple claims.

Basic JwtSecurityData configuration

The required portion of a JwtSecurityData is configured by the methods:

```
JwtSecData >> addAudience: audience;
JwtSecData >> addIssuer: issuer;
JwtSecData >> addUserId: userIdOfAUserProfile.
```

By default, the JwtSecurityData will look for the userId under the 'aud' in the JWT; this is only required if using a different JSON key for the UserId in the JWT.

For example:

```
aJwtSecData:= JwtSecurityData new.
aJwtSecData
   addAudience: 'GsAdminUser';
   addIssuer: 'https://accounts.google.com';
   userIdKey: 'aud';
   addUserId: 'GsAdminUser'.
```

JwtUserClaims

Your JWT will normally include additional user claims. These are optional, but if there are claims in the JWT that are not configured in the JwtSecurityData, login will fail. Claims in the JWT that are not of concern for GemStone authentication can be configured with a wildcard to allow the JWT validation on that claim to pass.

The class JwtUserClaim has been added to flexibly support user claims. Each claim has the following options:

`jsonKey`
: A Symbol matching the key for the claim in the JWT. This must exist in the JWT, or login will fail.

`jsonKind`
: A symbol describing the type of JSON object for the value of the key. Must be one of #String, #Number, #ArrayOfStrings, #ArrayOfNumbers, or #Boolean.

  If the jsonKind is set to #ArrayOfStrings or #ArrayOfNumbers, the JWT's value can be either an Array or a single value of String or Number.

`acceptedValues`
: an Array of one or more Symbols that includes all acceptable values. Each value must be a Symbol. If the JWT's value at #jsonKey is not in the list of values, login will fail. If the wildcard #* is included, any value will be accepted.

To add a claim, create it using class methods:

```
JwtUserClaim class >> newWildcardClaimWithJsonKey: keySym
   jsonKind: kindSym
JwtUserClaim class >> newWithJsonKey: keySym jsonKind: kindSym
   acceptedValues: anArray
```

And add the claim to the security Data using:

```
JwtSecurityData >> addClaim:
```

See the image methods for JwtUserClaim to see the protocol for further modifying the configuration and for further specifying wildcards.

For example:

```
claim := JwtUserClaim
   newWithJsonKey: #sub
   jsonKind: #String
   acceptedValues: { #'115156884418451143667' }.
aJwtSecData addUserClaim: claim.
```

Enable JWT Authentication in the UserProfile

To enable JWT authentication for a user, a new method has been added:

```
UserProfile >> enableJwtAuthenticationWith: aJwtSecurityData
```

Executing this method enables JWT authentication with the given *aJwtSecurityData*. A deep copy of *aJwtSecurityData* is made, and all elements securityPolicy is set to the SystemSecurityObjectSecurityPolicy.

System users, including SystemUser, DataCurator, GcUser, SymbolUser, Nameless, and HostAgentUser, cannot be configured to authenticate using JWTs.

Users configured with JWT Authentication can be disabled by administrators the same away as other non-GemStone-authentication accounts are managed.

The following methods allow you to fetch the security data and collect information about a UserProfiles's JWT authentication.

`UserProfile >> authenticationSchemeIsJWT`
Returns true if the receiver is configured to authenticate using a JWT.

`UserProfile >> authenticationScheme`
Returns #JWT for UserProfiles with JWT authentication.

`UserProfile >> authenticationScheme:` *aScheme*
This method now accepts #JWT to configure JWT authentication.

`UserProfile >> jwtSecurityData`
Returns a deep copy of the UserProfile's current JwtSecurityData, or nil if the receiver is not configuredd to authenticate with JWT.

`UserProfile >> validateJwtPassword:` *aJwtString*
Validates a JWT for login as the receiver. The receiver must have JWT authentication enabled. aJwtString must be a String which is a valid JsonWebToken in base64url format. Returns true if the JWT password is valid for the receiver. Otherwise returns a String describing an error condition.

Modifying a JwtSecurityData

You may make changes to an already-configured UserProfile's JwtSecurityData. Note that the act of setting and fetching both make copies, you do not modify the actual instance in use.

You may fetch a copy of the current JwtSecurityData using `aUserProfile jwtSecurityData`. This is a deep copy and can be edited freely. After modifying, invoke `enableJwtAuthenticationWith:` again with the updated JwtSecurityData, to update the UserProfile. This step also makes a deep copy, so futher edits will not affect the UserProfile.

See the image methods for JwtSecurityData to see the protocol for further modifying the configuration.

Example 2.1 Enable JWT Authentication for a UserProfile

Enable authentication for an existing user GsAdminUser,

```
| jwtSecData claim user |

aJwtSecData:= JwtSecurityData new.
aJwtSecData
   addAudience: 'GsAdminUser';
   addIssuer: 'https://accounts.google.com';
   userIdKey: 'aud';
   addUserId: 'GsAdminUser'.

claim := JwtUserClaim
   newWithJsonKey: #sub
   jsonKind: #String
   acceptedValues: { #'115156884418451143667' }.
aJwtSecData addUserClaim: claim.

claim := JwtUserClaim
   newWithJsonKey: #azp
   jsonKind: #String
   acceptedValues: { #'115156884418451143666' } .
aJwtSecData addUserClaim: claim.

claim := JwtUserClaim
   newWithJsonKey: #email
   jsonKind: #String
   acceptedValues: { #'service-account@oauth2-test-
      436914.iam.gserviceaccount.com' }.
aJwtSecData addUserClaim: claim.

claim := JwtUserClaim
   newWithJsonKey: #email_verified
   jsonKind: #Boolean
   acceptedValues: { true } .
aJwtSecData addUserClaim: claim.

(AllUsers userWithId: 'GsAdminUser')
   enableJwtAuthenticationWith: aJwtSecData.

System commit
```

2.3 Logging In

Password Source Locations

In 3.7.4, GemStone adds support for passwords to be located in environment variables or in disk files, in addition to being entered directly.

The environment variable name or disk file path (relative or absolute) is put into the login parameters password field (this parameter name is no longer entirely accurate, but retained for compatibility). When using a password in a disk file, the file should only include the password.

You must also set the appropriate login flag to indicate the password source location. how this is done depends on the client and is described in later sections.

JWT in JSON (JSON-wrapped JWT)

Passwords located in environment variables or on disk apply to most Authentication Scheme passwords (GemStone-native, UNIX, LDAP, and JWT), and can be used for login in all client environments (Topaz, GBS, GCI/GemBuilder for C, and external sessions).

For JWT logins only, there is an addition file-based option. It is allowed for the filename to refer to a file containing JSON, in which a key in the top level of the JSON refers to the JWT. Logins using JSON-wrapped JWT are available from Topaz and GBS only.

The JSON-wrapped JWT contents of this file might have the form, for example:

Example 2.2 JSON-wrapped JWT

```
os$> cat /gshost/gemstone/jwtlogin.jsonpass
{
"access_token":
      "eyJhbGciOiJSUzI1NiIsImtpZCI6ImRkMTI1ZDVmNDYyZmJjNjAxNGFl
      ZGFiODFkZGYzYmNlZGFiNzA4NDciLCJ0eXAiOiJKV1QifQ.eyJhdWQiOi
      JMaXNhIiwiYXpwIjoiMTE1MTU2ODg0NDE4NDUxMTQzNjY2IiwiZW1haWw
      iOiJzZXJ2aWNlLWFjY291bnRAb2F1dGgyLXRlc3QtNDM2OHE0LmlhbS5n
      c2VydmljZWFjY291bnQuY29tIiwiZW1haWxfdmVyaWZpZWQiOnRydWUsI
      mV4cCI6MTczNjgzNDk-3OCwiaWF0IjoxNzM2ODMxMzc4LCJpc3MiOiJod
      -HRwczovL2FjY291bnRzLmdvb2dsZS5jb20iLCJzdWIiOiIxMTUxNTY4O
      DQ0MTg0NTExNDM2NjYifQ.Ml0Cdx01k4Z1T5PoHdaioXE1Bab_e9SITXR
      Fj3tUVg09JtywU3X88ozhz3aKnUF_CT6e8ZSNjDqSjy9AlMjGEnxgMUVa
      0slIXM7wTg4jcoKikJ0uAadi6RKX9KZBLl0yBfc8xE92svWF6RV3tBeKi
      Uyu9CcIVsMQZrDMes2nf7zFoE3sNrh9x7AHlopS5UW5gm4SXJO3CDesIJ
      5jqkLDFuvkMPWZm8maXrYeDVWQHb5fspa4EZ4YPJVTwwFuLxQZ_CPfluS
      psaWHtymzfVyQU7Pj6oSFJ5UwFxFOLixsU_sK2GsnxLivxr9hotFkn8h8
      GjQ0OXj5bBtbU_7A",
"scope": "https://www.googleapis.com/auth/prediction",
"token_type": "Bearer",
"expires_in": 3600
}
```

When using a JSON-wrapped JWT to login, you must specify both the name of the file and the key that refers to the JWT.

Login via Topaz

Topaz includes additional **set** subcommands to configure login. For some additional details, see "Topaz added set subcommands" on page 13.

SET ENVPASSWORD *onOrOff*

Set to ON to treat password field as an envionrment variable containing a password.

SET FILEPASSWORD *onOrOff*

Set to ON to treat password field as the path to a disk file containing a password.

SET JWTJSONFILENAME *filename key*

Enables login using a JWT that is in a file with the name *filename*, which contain JSON. The JSON must include a key ***key*** that references the encoded JWT that to be used as the password for login. Executing this command makes topaz parse the JSON file and set the password to be the JWT. It also sets JWTPASSWORD to ON. May be abbreviated as JWTJ.

SET JWTPASSWORD *onOrOff*

When on, enables login using a JWT. A JWT password is valid for a specific user for a period of time until the JWT expires, and requires configuration of the Stone and UserProfile. May be abbreviated as JWTP or JWTPASS.

Examples of the Topaz JWT authentication options

1. To login with JWT encoded string

```
topaz> set jwtpass on
topaz> set user GsAdminUser
topaz> set password eyJhbGciOiJSUzI1NiIsImtpZCI6ImRkMTI1ZDVmNDY
   yZmJjNjAxNGFlZGFiODFkZGYzYmNlZGFiNzA4NDciLCJ0eXAiOiJKV1QifQ.
   eyJhdWQiOiJMaXNhIiwiYXpwIjoiMTE1MTU2ODg0NDE4NDUxMTQzNjY2Iiwi
   ZW1haWwiOiJzZXJ2aWNlLWFjY291bnRAb2F1dGgyLXRlc3QtNDM2OHE0Lmlh
   bS5nc2VydmljZWFjY291bnQuY29tIiwiZW1haWxfdmVyaWZpZWQiOnRydWUs
   ImV4cCI6MTczNjgzNDk3OCwiaWF0IjoxNzM2ODMxMzc4LCJpc3MiOiJod-HR
   wczovL2FjY291bnRzLmdvb2dsZS5jb20iLCJzdWIiOiIxMTUxNTY4ODQ0MTg
   0NTExNDM2NjYifQ.Ml0Cdx01k4Z1T5PoHdaioXE1Bab_e9SITXRFj3tUVg09
   JtywU3X88ozhz3aKnUF_CT6e8ZSNjDqSjy9AlMjGEnxgMUVa0slIXM7wTg4j
   coKikJ0uAadi6RKX9KZBLl0yBfc8xE92svWF6RV3tBeKiUyu9CcIVsMQZrDM
   es2nf7zFoE3sNrh9x7AHlopS5UW5gm4SXJO3CDesIJ5jqkLDFuvkMPWZm8ma
   XrYeDVWQHb5fspa4EZ4YPJVTwwFuLxQZ_CPfluSpsaWHtymzfVyQU7Pj6oSF
   J5UwFxFOLixsU_s-K2GsnxLivxr9hotFkn8h8GjQ0OXj5bBtbU_7A
topaz> login
```

2. To login with the encoded JWT in an environment variable

```
os$ > export GSAPASS=
   eyJhbGciOiJSUzI1NiIsImtpZCI6ImRkMTI1ZDVmNDYyZmJjNjAxNGFlZGFi
   ODFkZGYzYmNlZGFiNzA4NDciLCJ0eXAiOiJKV1QifQ.eyJhdWQiOiJMaXNhI
   iwiYXpwIjoiMTE1MTU2ODg0NDE4NDUxMTQzNjY2IiwiZW1haWwiOiJzZXJ2a
   WNlLWFjY291bnRAb2F1dGgyLXRlc3QtNDM2OHE0LmlhbS5nc2Vydmljc2WFjY
   291bnQuY29tIiwiZW1haWxfdmVyaWZpZWQiOnRydWUsImV4cCI6MTczNjgzN
   Dk3OCwiaWF0IjoxNzM2ODMxMzc4LCJpc3MiOiJod-HRwczovL2FjY291bnRz
   Lmdvb2dsZS5jb20iLCJzdWIiOiIxMTUxNTY4ODQ0MTg0NTExNDM2NjYifQ.M
   l0Cdx01k4Z1T5PoHdaioXE1Bab_e9SITXRFj3tUVg09JtywU3X88ozhz3aKn
   UF_CT6e8ZSNjDqSjy9AlMjGEnxgMUVa0slIXM7wTg4jcoKikJ0uAadi6RKX9
   KZBLl0yBfc8xE92svWF6RV3tBeKiUyu9CcIVsMQZrDMes2nf7zFoE3sNrh9x
   7AHlopS5UW5gm4SXJO3CDesIJ5jqkLDFuvkMPWZm8maXrYeDVWQHb5fspa4E
   Z4YPJVTwwFuLxQZ_CPfluSpsaWHtymzfVyQU7Pj6oSFJ5UwFxFOLixsU_s-K
   2GsnxLivxr9hotFkn8h8GjQ0OXj5bBtbU_7A
```

The topaz commands to login are:

```
topaz> set jwtpass on
topaz> set user GsAdminUser
topaz> set envpass on
topaz> set pass GSAPASS
topaz> login
```

3. To login using the encoded JWT in an file

The encoded password is located in a text file.

```
os$> cat /gshost/gemstone/jwtlogin.pass
   eyJhbGciOiJSUzI1NiIsImtpZCI6ImRkMTI1ZDVmNDYyZmJjNjAxNGFlZGFi
   ODFkZGYzYmNlZGFiNzA4NDciLCJ0eXAiOiJKV1QifQ.eyJhdWQiOiJMaXNhI
   iwiYXpwIjoiMTE1MTU2ODg0NDE4NDUxMTQzNjY2IiwiZW1haWwiOiJzZXJ2a
   WNlLWFjY291bnRAb2F1dGgyLXRlc3QtNDM2OHE0LmlhbS5nc2Vydmljc2WFjY
   291bnQuY29tIiwiZW1haWxfdmVyaWZpZWQiOnRydWUsImV4cCI6MTczNjgzN
   Dk-3OCwiaWF0IjoxNzM2ODMxMzc4LCJpc3MiOiJod-HRwczovL2FjY291bnR
   zLmdvb2dsZS5jb20iLCJzdWIiOiIxMTUxNTY4ODQ0MTg0NTExNDM2NjYifQ.
   Ml0Cdx01k4Z1T5PoHdaioXE1Bab_e9SITXRFj3tUVg09JtywU3X88ozhz3aK
   nUF_CT6e8ZSNjDqSjy9AlMjGEnxgMUVa0slIXM7wTg4jcoKikJ0uAadi6RKX
   9KZBLl0yBfc8xE92svWF6RV3tBeKiUyu9CcIVsMQZrDMes2nf7zFoE3sNrh9
   x7AHlopS5UW5gm4SXJO3CDesIJ5jqkLDFuvkMPWZm8maXrYeDVWQHb5fspa4
   EZ4YPJVTwwFuLxQZ_CPfluSpsaWHtymzfVyQU7Pj6oSFJ5UwFxFOLixsU_sK
   2GsnxLivxr9hotFkn8h8GjQ0OXj5bBtbU_7A
```

The topaz commands to login are:

```
topaz> set jwtpass on
topaz> set user GsAdminUser
topaz> set filepass on
topaz> set pass /gshost/gemstone/jwtlogin.pass
topaz> login
```

4. To login using the encoded JWT in JSON within a file

The encoded JWT should be in a key within JSON. For a file with contents such as described in Example 2.2 on page 42, the topaz commands would be:

```
topaz> set user GsAdminUser
topaz> set jwtjsonfilename /gshost/gemstone/jwtlogin.jsonpass
   access_token
topaz> login
```

Note that with the **jwtjsonfilename command,** you do not need to also use **set password** nor **set jwtpass**; these are set by **set jwtjsonfilename**.

Login via GsTsExternalSession

To login using JWT from an external session, the appropriate flags must be set on the session's parameters (an instance of GemStone Parameters).

Note that login using a JWT within a JSON file is not supported.

You must use GsTsExternalSession; JWT logins from GsExternalSession are not supported.

GemStoneParameters changed and added methods

Methods have been added to support the new password source locations as well as JWTs; see the full list of updated methods under "Additional Login Parameter Support" on page 11.

To login using JWTs, you can use the following combinations:

direct password

```
aGemStoneParameters jwtPassword: jwt
```

or

```
aGemStoneParameters setLoginWithJwt
aGemStoneParameters password: jwt
```

Password in disk file

```
aGemStoneParameters setLoginWithJwt
aGemStoneParameters passwordFileName: fileName
```

or

```
aGemStoneParameters setLoginWithJwt
aGemStoneParameters setPasswordIsFileName
aGemStoneParameters password: filename
```

Password in environment variable

```
aGemStoneParameters setLoginWithJwt
aGemStoneParameters passwordEnvVar: envVar
```

or

```
aGemStoneParameters setLoginWithJwt
aGemStoneParameters setPasswordIsEnvVar
aGemStoneParameters password: envVar
```

GsTsExternalSession added methods

Methods have been added to support the new password source locations as well as JWTs; see the full list of updated methods under "Additional Login Parameter Support" on page 11.

The following methods are available:

```
GsTsExternalSession >> jwtPassword: aString
GsTsExternalSession >> passwordEnvVar: aString
GsTsExternalSession >> passwordFileName: aString
```

Examples using GsTsExternalSession

Using the JWT directly

```
| sess res |
sess := GsTsExternalSession newDefault.
sess parameters
   username: 'GsAdminUser';
   setLoginWithJwt;
   password: 'eyJhbGciOiJSUzI1NiIsImtpZCI6ImRkMTI1ZDVmNDYyZmJjN
      jAxNGFlZGFiODFkZGYzYmNlZGFiNzA4NDciLCJ0eXAiOiJKV1QifQ.eyJ
      hdWQiOiJMaXNhQGdlbXRhbGtzeXN0ZW1zLmNvbSIsImF6cCI6IjExNTE1
      Njg4NDQxODQ1MTE0MzY2NiIsImVtYWlsIjoic2VydmljZS1hY2NvdW50Q
      G9hdXRoMi10ZXN0LTQzNjkxNC5pYW0uZ3NlcnZpY2VhY2NvdW50LmNvbS
      IsImVtYWlsX3ZlcmlmaWVkIjp0cnVlLCJleHAiOjE3MzY5NzU3MTMsIml
      hdCI6MTczNjk3MjExMywiaXNzIjoiaHR0cHM6Ly9hY-2NvdW50cy5nb29
      nbGUuY29tIiwic3ViIjoiMTE1MTU2ODg0NDE4NDUxMTQzNjY2In0.i60x
      I27dL2OpjCCw_nC0ZnBYcMvGpIXtYqhubzwLA1qfYd40tcMpOsMK28OVC
      Zr3g7PkzQO31gbhPMlKUsT0fHKT30vbYRcuCHXo9gjskn2w_qUgn1y7IV
      Khv0ixvCbsbilhwoHI0gMmnFInQK85iFqPPswe914_xAq26-ILq0NJKvT
      gJXUGiurCq0EOruONio8QNLKbSoxzVwBdxXJ63V0LzWx1wLy4SpR63SKO
      hRqcXtRDcMIzQTWhJ4BUnsueJW5SXidWy10V_4UINC1sEjlqDRdCox1BZ
      uEFZkwmDzRAzCmY7T4W6XozFkigljskei_RfekqW090pe52qunOw'.
sess login.
```

JWT in an environment value

With the JWT in an environment variable as it is defined as "To login with the encoded JWT in an environment variable" on page 44:

```
| sess res |
sess := GsTsExternalSession newDefault.
sess parameters
   username: 'GsAdminUser';
   setLoginWithJwt;
   passwordEnvVar: 'GSAPASS'.
sess login.
```

JWT in an disk file value

With the JWT in an disk file, as described under "To login using the encoded JWT in an file" on page 44:

```
| sess res |
sess := GsTsExternalSession newDefault.
sess parameters
   username: 'GsAdminUser';
   setLoginWithJwt;
   passwordFileName: '/gshost/gemstone/jwtlogin.pass'.
sess login.
```

Login via GemBuilder for C

Login flags have been to support JWT logins, and logins via other authentication schemes with password in file or environment variable. These flags apply to both classic GCI logins using GciLoginEx() and related methods, and thread-safe GCI logins using GsTsLogin() and related methods.

There is no direct support for login with files containing a JWT in JSON.

See "Added login flags" on page 23 for specifics on the new flags.

2.4 JsonWebToken

The class JsonWebToken has been added. Instances of JsonWebToken provide an easier way of manipulating the JSON of a JWT than using JSON itself, and JsonWebToken includes protocol to encode and decode JWTs. JsonWebToken is intended to be a general purpose class to support other JWT use cases.

Use of JsonWebToken is not needed to configure authentication using regularly provided JWT whose public keys are managed by OpenId discovery.

JsonWebToken can be used to create temporary JWTs, with public keys that are manually added to the Stone. This is a convenience for testing JWT login configuration and authentication. Manually added public keys are temporary and not retained over Stone shutdown.

Creating and updating a JsonWebToken

Creating a JsonWebToken

New JsonWebTokens for JWT authentication are created using:

```
JsonWebToken newForRsa256
```

While the JsonWebToken class supports JWTs signed using Hash-Based Method Authentication Codes (HMACs) with secret keys, these are not supported for logging into GemStone. See the image for methods supporting non-login related JWTs.

Converting JWTs

JWTs can be converted between the JsonWebToken object format, and the encoded string format or the unencoded JSON. However the JsonWebToken was created, it can be edited as needed, and encoded or re-encoded into a JWT String or printed in JSON.

To create a `JsonWebToken` from an encoded JWT String:

```
JsonWebToken fromJwtStrig: aJwtString
```

To print the JSON for a JsonWebToken:

```
JsonWebToken asJson
```

To create an encoded JWT string from a JsonWebToken:

```
JsonWebToken asJwtString
```

Updating a JsonWebToken

There are a number of methods to access and update the fields of a JsonWebToken, and several specific test methods.

```
JsonWebToken >> algorithm, JsonWebToken >> algorithm:
JsonWebToken >> audience, JsonWebToken >> audience:
JsonWebToken >> authorizedParty, JsonWebToken >> authorizedParty:
JsonWebToken >> expirationTime, JsonWebToken >> expirationTime:
JsonWebToken >> issuedAtTime, JsonWebToken >> issuedAtTime:
JsonWebToken >> issuer, JsonWebToken >> issuer:
JsonWebToken >> keyId, JsonWebToken >> keyId:
JsonWebToken >> type, JsonWebToken >> type:
JsonWebToken >> subject, JsonWebToken >> subject:
JsonWebToken >> notBeforeTime, JsonWebToken >> notBeforeTime:
JsonWebToken >> payloadClaimAt:
JsonWebToken >> payloadClaimAt:put:
JsonWebToken >> headerClaimAt:
JsonWebToken >> headerClaimAt:put:
JsonWebToken >> allPayloadClaims
JsonWebToken >> allHeaderClaims
JsonWebToken >> allClaims
JsonWebToken >> secondsUntilExpiration
JsonWebToken >> isExpired
JsonWebToken >> isSigned
```

After the JsonWebToken is updated, it must be signed with a private key, using:

```
JsonWebToken >> signWithPrivateKey: privKey.
```

After it is signed, you should not make any further changes. Updates to the JsonWebToken are ignored after signing.

Using a JsonWebToken to verify login

Normally, JWTs are generated and signed by external services, and validated within GemStone using the public key against an OpenId discovery document. A JsonWebToken is not involved.

For testing, GemStone supports manually adding a public key to the Stone, which allows you to generate a JWT, sign it, and use that JWT for authentication. This allows you to test JWTs that do not exactly match the format generated by your JWT service.

The keys manually added to the Stone are not persistent over Stone restart.

Manual management of public keys in Stone

The following methods support manually adding and removing JWT public keys from the Stone's set of temporary keys.

`System class >> addJwtKey:` *tlsObj* `withId:` *keyIdString*
: Adds *tlsObj* to the list of temporary public keys used to authenticate JWT logins. *tlsObj* must be an instance of either GsX509Certificate or GsTlsPublicKey. Only RSA public keys and certificates are supported. *keyIdString* must be an instance of String with a length of 1 or more characters. Keys added using this method are temporary and are lost when the Stone is restarted.

`System class >> removeJwtKeyWithId:` *keyIdString*
: Removes the temporary key with *keyIdString* from the list of public keys used to authenticate JWT logins. *keyIdString* must be an instance of String with a length of 1 or more characters. Only keys that were manually added using #addJwtKey:withId: can be removed; keys added by the OID thread cannot be removed.

`System class >> removeAllJwtKeys`
: Removes all temporary keys from the list of public keys used to authenticate JWT logins. Only keys previously added using the #addJwtKey:withId: method are removed. Removing keys not added by the above method is not supported. It is not an error if there are no keys eligible for removal.

`System class >> removeJwtKeyWithId:` *keyIdString* `ifAbsent:` *aBlock*
: Removes the temporary key with *keyId* from the list of temporary public keys used to authenticate JWT logins. If the key is not found, returns the result of evaluating the zero-argument block aBlock. *keyIdString* must be an instance of String with a length of 1 or more characters. The key with *keyIdString* must have been previously added using the #addJwtKey:withId: method. Removing keys not added by the above method is not supported.

Example

The following example demonstrates how to create a JWT and create a corresponding JwtSecurityData, and configure a user to authenticate with that JwtSecurityData.

By manually loading the public key for the key id into the Stone, there is no need for verification of a valid key nor does the server need to be configured for the discovery id.

Example 2.3 Login using JsonWebToken

```
| kid userName privKey userPro jwt jsd now sess |

kid := 'C2C587F2718953AE8A4B37307C1641F3'.
userName := 'GsAdminUser'.
userPro := AllUsers userWithId: userName ifAbsent:[
   userPro := AllUsers
      addNewUserWithId: userName password: 'swordfish'.
   System commit. userPro ].

jsd := JwtSecurityData new.
jsd
   addAudience: userName;
   addIssuer:  'https://test.gemtalksystems.com';
   addUserId: userName;
   addUserClaim: (JwtUserClaim
      newWithJsonKey: #azp
      jsonKind: #String
      acceptedValues: { '123456789'}) .
userPro enableJwtAuthenticationWith: jsd.
System commit.

privKey := GsTlsPrivateKey
   newFromPemFile: '$GEMSTONE/examples/openssl/private/backup_s
      ign_1_clientkey.pem'
   withPassphraseFile: '$GEMSTONE/examples/openssl/private/back
      up_sign_1_client_passwd.txt'.

jwt := JsonWebToken newForRsa256.
jwt audience: userName;
   issuer: 'https://test.gemtalksystems.com';
   keyId: kid;
   authorizedParty: '123456789';
   issuedAtTime: (now := System timeGmt);
   expirationTime: now + 7200;
   signWithPrivateKey: privKey.
jwt := jwt asJwtString.

System addJwtKey: (privKey asPublicKey) withId: kid.

sess :=  GsTsExternalSession newDefault .
sess
   username: userPro userId ;
   jwtPassword: jwt.
sess login.
```

You should ensure that the public keys are removed f rom the Stone's cache when they are no longer needed.

```
topaz 1> run
   System jwtPublicKeys
%
[52089345 size:2  Array] anArray
  #1 [52088577 size:32  String]
   C2C587F2718953AE8A4B37307C1641F3
  #2 [52088065  GsTlsPublicKey] aGsTlsPublicKey

topaz 1> run
   System removeJwtKeyWithId:
      'C2C587F2718953AE8A4B37307C1641F3'.
%
```

2.5 Login Tracking

The Login Log, if enabled, records JWT successful and failed logins and logouts, as for any login.

The Login Log output now has an additional column providing the specific authentication scheme used for that login.

The options for Login authentication kind are:

0 - GemStone password
1 - UNIX/PAM
2 - LDAP
3 - Kerberos/SSO
4 - X509
5 - JSON Web Token (JWT)

2.6 Debugging

There are many moving parts to a JWT login. Login debugging has been added that reports the specific validations tests and a failure that prevented login, if login failed.

When login debugging is enabled, the JWT validations are printed to stdout; the linked console, or the Gem log file. This allows you to see which validation failed on a login failure.

Following are examples of some of the validations that are printed:

```
[Debug]: Timestamp is valid
[Debug]: JWT issuer https://accounts.google.com is authorized
[Debug]: Checking user claim sub: valid
```

Login debugging is printed for all logins, not just JWT logins, but other logins do not provide any further detail. For example:

```
[Debug]: UserProfile auth kind is: Password (GemStone)
[Debug]: Login failed:  the userId/password combination is
   invalid or expired.; badPassword
```

Login debugging can be configured in a number of ways.

Environment variable GS_DEBUG_LOGIN

To help determine the cause of login failures, set the environment variable GS_DEBUG_LOGIN to any value, in the environment of the client. This sets the bit in the login parameters, which can also be done as described in the following sections.

GBS login debugging

To debug JWT logins on GBS, use the GS_DEBUG_LOGIN environment variable.

GCI flags

To enabled login debugging in a GCI login, use the new login flag GCI_LOGIN_DEBUG. See“Added login flags for GciLoginEx() and GsTsLogin()” on page 23.

Image methods

When logging in via an external session, configure the GsTsExternalSession or GemStoneParameters to include the debugging flag, using either of the following methods.

`GsTsExternalSession >> setLoginDebug`
Enables writing debug info to the gem's log file.

`GemStoneParameters >> setLoginDebug`
Enable writing login debug details to the gem's log file.