

Geodesic Systems, Inc.
© Copyright 1994-2002



User's Guide

Version 6.0

Table of Contents

INTRODUCTION	3
Contacting Geodesic	3
Using this Guide	5
GETTING STARTED	7
System Requirements	7
Using Runtime: A Quick Reference.....	9
For Great Circle Users	9
Installation	9
The Geodesic Runtime Files.....	10
Enabling Geodesic Runtime	11
The Geodesic Runtime Product License.....	12
PLATFORM SPECIFIC INFORMATION	13
HP-UX	14
IBM AIX.....	17
Linux.....	21
Microsoft Windows.....	24
Sun Solaris.....	28
TUNING RUNTIME'S PERFORMANCE	32
Definitions	32
When to Tune Runtime's Performance.....	32
When Not to Tune Runtime's Performance	33
How to Tune Runtime's Performance	33
Controlling When Garbage Collection Occurs	34
Tuning Runtime's Virtual Memory Performance	35
Optimization for Speed	36
Multi-Threading	37
TUNING RUNTIME'S RELIABILITY	39
Optimization for Reliability.....	39
Automatically Fixing Premature Frees.....	40
ADMINISTRATOR'S GUIDE	42
Configuring Geodesic Runtime	42
Geodesic Runtime Environment Variables.....	43
Deprecation Information.....	52
Geodesic Runtime Log File.....	53
TROUBLESHOOTING GUIDE	54

Troubleshooting Injecting with Geodesic Runtime.....	54
Troubleshooting Running With Geodesic Runtime	55
APPENDIX	57
Introduction to Memory Management.....	57
The Geodesic Product Family.....	61
GLOSSARY	63
NOTICES	65
Proprietary Rights.....	65
INDEX	66

Introduction

CONTACTING GEODESIC

We are genuinely committed to your satisfaction—backing all our products with a dedicated support operation. We invite you to contact Geodesic with your valuable feedback: the more feedback you give us, the more responsive to your needs we can be. We back our products with a dedicated support operation. We also want to inform you about our latest products, platforms, and updates as they become available.

For Sales, Registration, Product Information, or Comments

Phone: +1 312-832-1221 (Toll-free in the USA 800-360-8388)
Fax: +1 312-832-1230
E-Mail: info@geodesic.com
WWW: <http://www.geodesic.com>
Address: Geodesic Systems
414 N. Orleans St., Suite 410
Chicago, IL 60610, USA

For Technical Support

Geodesic provides technical support by phone, e-mail and fax. See our support policy for further details (<http://www.geodesic.com/support/support.html>). Reasonably priced, annual support contracts - including all updates - are available from Geodesic.

Phone: +1 312-923-8710
Fax: +1 312-832-1230
E-Mail: support@geodesic.com
WWW: <http://www.geodesic.com>
Address: Geodesic Systems
ATTN: Technical Support
414 N. Orleans St., Suite 410
Chicago, IL 60610, USA

To make your technical support call go faster, please have the following information ready:

- 1) Your company and name.
- 2) Operating system name and version.
- 3) Compiler name and version.
- 4) The Runtime product and version you are running.
You'll find this in the first four lines of the `readme.txt` file located in the Runtime install directory.
- 5) A description of the problem you are experiencing.
For example, if you program crashes, what sort of run-time error occurred? Is a stack trace available? Was any message sent to `stderr`?
- 6) Any log file produced (usually `gs*.log` in the temp directory or wherever it was redirected with `GS_LOG_FILE_PATH`).
- 7) A list of all Runtime-related environment variables that have been set and their values.
- 8) A list of all Runtime API calls being made by your program, including the parameters being passed to each.

USING THIS GUIDE

Throughout the manual, certain typefaces and icons are used to make the information specific to your needs pop out easily.

Typefaces

Most of the text is in the font of this sentence.

Programming language, filenames, command line text, function calls or parameters of a function are shown in the font of this sentence.

Icons

Operating Systems

The following icons are used to denote platform specific comments.



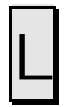
Unix



HP-UX



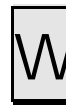
IBM AIX



Linux



Sun Solaris



Microsoft
Windows

Notes



Note – just a basic note to the user.



C++ - the comment is for C++ programs only.



Warning – take heed to the comment.



SMP – this comment or function relates to the SMP product only.



Advanced – this comment or function is for advanced users.

Getting Started

SYSTEM REQUIREMENTS

Geodesic Runtime currently supports the following operating systems and compilers.

Architecture	Operating System	Compiler
HP-PA-RISC	HP-UX v11	HPa1/HPaC++
HP-PA-RISC	HP-UX v11i	HPa1/HPaC++
IBM RS6000	AIX 4.3.3	Cset 3.6.6
IBM RS6000	AIX 4.3.3	VisualAge 5.0 Prof
IBM RS6000	AIX 5L	VisualAge 5.0 Prof
Intel Itanium	Linux 64-bit, kernel 2.4.3-2.10.1smp, glibc 2.2.3-10 (RedHat 7.0.98)	GNU 3.0
Intel Pentium	Linux kernel 2.2.14-5.0smp, glibc 2.1.3-15 (RedHat 6.2)	GNU 2.95.x +
Intel Pentium	Linux kernel 2.2.16-22 with glibc 2.1.3-15 (RedHat 7.0)	GNU 2.95.x +
Intel Pentium	Linux kernel 2.4.2-2smp with glibc 2.2.2.10 (RedHat 7.1)	GNU 2.95.x +
Intel Pentium	MS Windows 2000	MS Visual C++ V6.0
Intel Pentium	MS Windows NT 4.0. SP 3	MS Visual C++ V6.0
Intel Pentium	MS Windows XP	MS Visual C++ V6.0
Sun SPARC	Solaris 2.6	Forte 6.0
Sun SPARC	Solaris 2.6	GNU 2.95.x +
Sun SPARC	Solaris 2.6	Visual Workshop 4.2
Sun SPARC	Solaris 2.6	Visual Workshop 5.0
Sun SPARC	Solaris 7	Forte 6.0
Sun SPARC	Solaris 7	GNU 2.95.x +
Sun SPARC	Solaris 7	Visual Workshop 4.2

Architecture	Operating System	Compiler
Sun SPARC	Solaris 7	Visual Workshop 5.0
Sun SPARC	Solaris 8	Forte 6.0
Sun SPARC	Solaris 8	GNU 2.95.x +
Sun SPARC	Solaris 8	Visual Workshop 4.2
Sun SPARC	Solaris 8	Visual Workshop 5.0

USING RUNTIME: A QUICK REFERENCE

To use Runtime, follow these four steps. You will find further details for each step in the “Platform Specific Information” section for your operating system, starting on page 13:

- 1) Install Runtime as described in the installation section specific to your platform.
- 2) Determine which library you want to use. See the list of libraries and object files for your platform.
- 3) Incorporate the Runtime license described in “The Geodesic Runtime Product License” on page 12.
- 4) Inject Runtime into your application.

FOR GREAT CIRCLE USERS

Runtime and Great Circle are both memory allocators and built on similar technology. It is not possible to use both at the same time. One will either override the other or they will interfere with each other and cause problems.

When to use Runtime: In production environment. It remediates memory problems while offering a performance benefit from its fast allocator (particularly on SMP machines).

When to use Great Circle: During development and testing. It helps to debug memory problems, showing leaks, overwrites, wild frees, etc. Great Circle is not appropriate for a production environment because the performance hit for all the symbol reading can be enormous.

INSTALLATION

The basic steps to install Runtime on your computer are as follows:

- 1) Read the appropriate platform specific instructions for your operating system.
 - For HP-UX, see page 14.
 - For IBM AIX, see page 17.
 - For Linux, see page 21.
 - For Microsoft Windows, see page 24.
 - For Sun Solaris, see page 28.
- 2) Choose a file system with at least 6 MB free.
- 3) Install the Runtime files.

THE GEODESIC RUNTIME FILES

Directories

The default root directory for the Runtime files depends upon your operating system and the version of Runtime installed. Refer to the platform specific section for your operating system for details.

For HP-UX, see page 14.

For IBM AIX, see page 17.

For Linux, see page 21.

For Microsoft Windows, see page 24.

For Sun Solaris, see page 28.

Libraries and Object Files

The platform section for each operating system contains a guide to the files distributed for every platform and compiler.

ENABLING GEODESIC RUNTIME

There are two methods to use Runtime with your application: injecting and linking. Should you wish to directly work with the source code, please refer to the programmer's reference for information.

Injecting Runtime into Your Program

The injector is a program that permits you to use the Runtime libraries with existing executables, without any need to re-link. The methods of injection vary by operating system. Injection replaces the default memory subsystem with our more efficient subsystem. This does not require recompiling, but rather dynamically loads or redirects memory calls to our Runtime libraries instead.

Injection Instructions

For details, see your specific platform instructions:

For HP-UX, see page 15.

For IBM AIX, see page 18.

For Linux, see page 22.

For Microsoft Windows, see page 25.

For Sun Solaris, see page 29.

THE GEODESIC RUNTIME PRODUCT LICENSE

Runtime product licenses come in two different forms: Text licenses and C Source licenses. Should you be planning to link with Runtime, see the programmer's reference for use of the C Source license.

Text Licenses

You can place this file anywhere you choose, but you must register its name and location using the `GS_LICENSE_FILE` environment variable on page 47.



If you do not set this variable, Runtime will look for `gslicense.txt` in the system directory (e.g. `c:\winnt\system32`). For information on setting environment variables, see the section “Setting Windows Environment Variables” on page 27.



On Unix, except for HP-UX, when using injection, the `-c` option can also be used to specify the location of the text license file. See the appropriate appendix for your operating system for more information.

Platform Specific Information

This section contains instructions and information specific to each operating system we support. Specifically, you'll find out how to install and inject Runtime.

- HP-UX, page 14
- IBM AIX, page 17
- Linux, page 21
- Microsoft Windows, page 24
- Sun Solaris, page 28

HP-UX

Runtime currently supports the HP-UX v11/v11i operating system with the HPa1/HPaC++ compiler.



Footprint reduction is not supported on the HP-UX platform.

Installation

Depending on where the desired installation directory is located, root access may be required for installation. For a description of the files that are installed, see “The Runtime Directories” on page 14.

- 1) Copy the tar file distribution to the desired installation directory.
- 2) Uncompress the tar file.

```
uncompress <filename>.tar.Z
```

- 3) Unpack the tar file. Runtime will be unpacked into its own subdirectory.

```
tar -xf <filename>.tar
```



Product

<filename>

High Availability	ha_runtime_up
High Availability SMP	ha_runtime_smp

- 4) It is now safe to remove the tar file.
- 5) To use the injector, run `inject-setup`. You must be logged in as ‘root’ to run the injector setup script. Change to the directory `<ROOT_DIR>/etc/inject` and run the injector setup script `inject-setup`. The script creates a `/GS/lib` directory and creates symbolic links to the Runtime libraries there.

The Runtime Directories

The root directory depends upon where the files were unpacked and which Runtime product was installed.

Product	Root Directory
High Availability	<UNPACK-DIR>/ha_runtime_up/<VERSION>/hp11/pa/hpa
High Availability SMP	<UNPACK-DIR>/ha_runtime_smp/<VERSION>/hp11/pa/hpa

<UNPACK-DIR> is the directory in which the unpack command for the tar file was issued. See “Installation” on page 14.

<VERSION> is the current released version of Runtime, for example, 6.0.1.5.

The root directory will be referred to as <ROOT_DIR>. This directory contains:

- The readme, release notes, and installation files.
- A `bin/` directory with the injector.
- A `doc/` directory with the manual.
- An `etc/` directory containing any extra configuration scripts.
- An `include/` directory containing C header files.
- A `lib/` directory containing the libraries and object files.
- A `man/` directory containing the man pages for `gsinject`.

Injecting Runtime

Under HP-UX, it is necessary to produce an injected version of an application with `gsinject` and then run the newly injected program with whatever arguments are required. Prior to using the injector for the first time, be sure that you’ve run `inject-setup` as described in the “Installation” section on page 14.

1. Choose an application to inject.
2. Set the flag for the High Availability Runtime library.
3. Choose whether you want to inject a single- or multi-threaded library. It is always safe to use the multi-threaded or SMP library. It is only safe to use the single-threaded library if the application is single-threaded.
4. Run `gsinject` using the parameters described below. For example, to inject the High Availability Runtime multi-threaded library into `myprog`, use the command:

```
gsinject -t -a myprog myprog-injected
```

5. Make sure that the library can find a valid Runtime text license (using the `GS_LICENSE_FILE` environment variable). See “The Geodesic Runtime Product License” on page 12 for more details.
6. Run the injected binary.

Syntax of the *gsinject* command

```
gsinject (-s|t|p) -a [-v] [-h] in out
```

Threading flags:

- s** Injects single-threaded flavor of the library.
- t** Injects multi-threaded flavor of the library.
- p** Injects SMP flavor of the library.

Library selection flags:

- a** Injects High Availability Runtime library.

Other options:

- v** Verbose mode. Messages about injection will be displayed on screen.
- h** Displays the help text.

In - Out

- in** Name of the executable to be injected.
- out** Name of the injected copy.

IBM AIX

Runtime currently supports the following:

Architecture	Operating System	Compiler
IBM RS6000	AIX 4.3.3	Cset 3.6.6
IBM RS6000	AIX 4.3.3	VisualAge 5.0 Prof
IBM RS6000	AIX 5L	VisualAge 5.0 Prof

Installation

Depending on where the desired installation directory is located, root access may be required for installation. For a description of the files that are installed, see “The Runtime Directories”.

- 1) Copy the tar file distribution to the desired installation directory.
- 2) Uncompress the tar file.

```
uncompress <filename>.tar.Z
```

- 3) Unpack the tar file. Runtime will be unpacked into its own subdirectory.

```
tar -xf <filename>.tar
```



Product

<filename>

High Availability	ha_runtime_up
High Availability SMP	ha_runtime_smp

- 4) It is now safe to remove the tar file.

The Runtime Directories

The root directory depends upon the version of AIX and which Runtime product was installed.

AIX Ver.	Comp.	Product	Root Directory
4.3	VA5 or Cset	High Availability	<UNPACKDIR>/ha_runtime_up (<VERSION>/aix43/mc64/mc5

AIX Ver.	Comp.	Product	Root Directory
	or Cset	Availability	/<VERSION>/aix43/rs6k/va5
4.3	VA5 or Cset	High Availability SMP	<UNPACKDIR>/ha_runtime_smp /<VERSION>/aix43/rs6k/va5
5L	VA5	High Availability	<UNPACKDIR>/ha_runtime_up /<VERSION>/aix51/rs6k/va5
5L	VA5	High Availability SMP	<UNPACKDIR>/ha_runtime_smp /<VERSION>/aix51/rs6k/va5

<UNPACK-DIR> is the directory in which the unpack command for the tar file was issued. See “Installation” on page 17.

<VERSION> is the current released version of Runtime, for example, 6.0.1.5.

The root directory will be referred to as <ROOT_DIR>. This directory contains:

- The readme, release notes, and installation files.
- A `bin/` directory with the injector.
- A `doc/` directory with the manual.
- An `etc/` directory containing any extra configuration scripts.
- An `include/` directory containing C header files.
- A `lib/` directory containing the libraries and object files.
- A `man/` directory containing the man pages for `gsinject`.

Injecting Runtime

Under IBM AIX, injection is part of the way an application is invoked. Whereas normally an application is invoked as a command with arguments, with injection the `gsinject` command must be used to launch the application as described below.

1. Choose an application to inject. You will need to know the command and arguments normally used to launch it.
2. Set the flag for the High Availability Runtime library.
3. Choose whether you want to inject a single- or multi-threaded library. It is always safe to use the multi-threaded or SMP library. It is only safe to use the single-threaded library if the application is single-threaded.
4. Decide whether child processes launched by the application should also be injected. If the child process is a different application than the parent, keep in mind that Runtime

- may not be compatible with the child. Forked processes are always injected. If a process is forked and then `execed`, it will only be injected if the `-e` flag is used.
5. Make sure that the library can find a valid Runtime text license (using either `GS_LICENSE_FILE` or the `-c` option). See “The Geodesic Runtime Product License” on page 12 for more details.
 6. Run `gsinject` using the parameters described below. For example, to inject the High Availability Runtime multi-threaded library into `myprog` and all child processes, use the command:

```
gsinject -t -a -e myprog arg1 arg2
```

Syntax of the `gsinject` command

```
gsinject (-s|t|p) -a [-n|e] [-v] [-h] [-c license-file]  
command [arguments]
```

Threading flags:

- s** Injects single-threaded flavor of the library.
- t** Injects multi-threaded flavor of the library.
- p** Injects SMP flavor of the library.

Library selection flags:

- a** Injects High Availability Runtime library.

Exec behavior flags:

- n** (Default) If the application launches child processes using `exec`, they will not be injected.
- e** Child processes launched via `exec` will be injected.

Other options:

- v** Verbose mode. Messages about injection will be emitted to standard output.
- h** Displays help text.
- c** Specifies the location of the text license file. Overrides the `GS_LICENSE_FILE` variable.



If the application is multi-threaded, the environment variable `AIXTHREAD_SCOPE` must be set to `S`.



Footprint reduction will only work if `GS_USE_MMAP` is set to 1. At this time, this will reduce the speed of your application.



If an application has the “set UID” (set-user-ID-on-execution) bit set in its permissions, injection may not work because the application may not be able to find the Geodesic shared libraries. There are 2 ways around this:

1. When the program is compiled, add the directory of our libraries to its library path.

```
"-L <path to geodesic  
product>/6.0.0.9/aix43/rs6k/va5/lib"
```

2. Copy the Geodesic shared libraries (`lib*_*.a` in the products lib directory) into a directory that already exists in your program's library path. To check the library path of an executable you can use:

```
$> dump -H <executable name>
```

The library path should appear near the top of the “***Import File Strings***” section.

LINUX

Runtime currently supports the following:

Architecture	Operating System	Compiler
Intel Itanium	Linux 64-bit, kernel 2.4.3-2.10.1smp, glibc 2.2.3-10 (RedHat 7.0.98)	GNU 3.0
Intel Pentium	Linux kernel 2.2.14-5.0smp, glibc 2.1.3-15 (RedHat 6.2)	GNU 2.95.x +
Intel Pentium	Linux kernel 2.2.16-22 with glibc 2.1.3-15 (RedHat 7.0)	GNU 2.95.x +
Intel Pentium	Linux kernel 2.4.2-2smp with glibc 2.2.2.10 (RedHat 7.1)	GNU 2.95.x +

Installation

Depending on where the desired installation directory is located, root access may be required for installation. For a description of the files that are installed, see “The Runtime Directories”.

- 1) Copy the compressed tar file to the desired installation directory.
- 2) Uncompress and unpack the tar file. Runtime will be unpacked into its own subdirectory.

```
tar zxf <filename>.tar.Z
```



Product

<filename>

High Availability	ha_runtime_up
High Availability SMP	ha_runtime_smp

- 3) It is now safe to remove the compressed tar file.

The Runtime Directories

The root directory depends upon which Runtime product was installed.

Architecture	Product	Root Directory
Intel Itanium	High Availability	<UNPACK-DIR>/ha_runtime_up/<VERSION>/linux/ia64/gnu

Architecture	Product	Root Directory
Intel Itanium	High Availability SMP	<UNPACK-DIR>/ha_runtime_smp/<VERSION>/linux/ia64/gnu
Intel Pentium	High Availability	<UNPACK-DIR>/ha_runtime_up/<VERSION>/linux/x86/gnu
Intel Pentium	High Availability SMP	<UNPACK-DIR>/ha_runtime_smp/<VERSION>/linux/x86/gnu

<UNPACK-DIR> is the directory in which the unpack command for the tar file was issued. See “Installation” on page 21.

<VERSION> is the current released version of Runtime, for example, 6.0.1.5.

The root directory will be referred to as <ROOT_DIR>. This directory contains:

- The readme, release notes, and installation files.
- A `bin/` directory with the injector.
- A `doc/` directory with the manual.
- An `include/` directory containing C header files.
- A `lib/` directory containing the libraries and object files.
- A `man/` directory with any man pages.

Injecting Runtime

Under Linux, injection is part of the way an application is invoked. Whereas normally an application is invoked as a command with arguments, with injection the `gsinject` command must be used to launch the application as described below.

- 1) Choose an application to inject. You will need to know the command and arguments normally used to launch it.
- 2) Set the flag for the High Availability Runtime library.
- 3) Choose whether you want to inject a single- or multi-threaded library. It is always safe to use the multi-threaded or SMP library. It is only safe to use the single-threaded library if the application is single-threaded.
- 4) Decide whether child processes launched by the application should also be injected. If the child process is a different application than the parent, keep in mind that Runtime may not be compatible with the child. Forked processes are always injected. If a process is forked and then `execed`, it will only be injected if the `-e` flag is used.

- 5) Make sure that the library can find a valid Runtime text license (using either `GS_LICENSE_FILE` or the `-c` option). See “The Geodesic Runtime Product License” on page 12 for more details.
- 6) Run `gsinject` using the parameters described below. For example, to inject the High Availability Runtime multi-threaded library into `myprog` and all child processes, use the command:

```
gsinject -t -a -e myprog arg1 arg2
```

Syntax of the `gsinject` command

```
gsinject (-s|t|p) -a [-n|e] [-v] [-h] [-c license-file]  
command [arguments]
```

Threading flags:

- s** Injects single-threaded flavor of the library.
- t** Injects multi-threaded flavor of the library.
- p** Injects SMP flavor of the library.

Library selection flags:

- a** Injects High Availability Runtime library.

Exec behavior flags:

- n** (Default) If the application launches child processes using `exec`, they will not be injected.
- e** Child processes launched via `exec` will be injected.

Other options:

- v** Verbose mode. Messages about injection will be emitted to standard output.
- h** Displays help text.
- c** Specifies the location of the text license file. Overrides the `GS_LICENSE_FILE` variable.

MICROSOFT WINDOWS

Runtime currently supports the following:

Architecture	Operating System	Compiler
Intel Pentium	MS Windows 2000	MS Visual C++ V6.0
Intel Pentium	MS Windows NT 4.0. SP 3	MS Visual C++ V6.0
Intel Pentium	MS Windows XP	MS Visual C++ V6.0

Installation

You will need administrator privileges for installation. See “The Runtime Directories” for a description of the files that are installed.

- 1) Insert the Runtime CD into the CD-ROM device.
- 2) If Auto-Play is enabled, then the DemoShield interface will appear. If Auto-Play is disabled, click on `setup.exe` in the root of the CD-ROM directory and this will launch DemoShield.
- 3) From the DemoShield interface, click “Install Runtime” to launch the InstallShield installer.
- 4) From the InstallShield installer, you will be prompted with a series of dialogs that will request such information as “where to install the product”. Follow the dialogs until the installation is complete.
- 5) Reboot your system.

The Runtime Directories

The root directory depends upon which product was installed.

Product	Default Root Directory
High Availability	C:\Program Files\Geodesic Systems\ Runtime High Availability <VERSION>
High Availability SMP	C:\Program Files\Geodesic Systems\ Runtime High Availability SMP <VERSION>

<VERSION> is the current released version of Runtime, for example, 6.0.1.5.

The root directory will be referred to as <ROOT_DIR>. This directory contains:

- The readme, release notes, and installation files.
- A `doc/` directory containing the manual.
- An `include/` directory containing C header files.

- An `msvc6/` directory containing the libraries and object files.
- A `mapfiles/` directory containing the linker files Runtime uses for troubleshooting. In case of problems, Geodesic Systems Technical Support may ask you to look at these files.



On Windows systems, the Runtime DLLs are copied to your computer's system directory as appropriate for your configuration; this includes `gsinjdll.dll`, which is needed for injection. Also, Runtime may install `msvcrt.dll` if you do not have the most recent version.

DLL Files

(shared libraries only)

Product	Single-Threaded	Multi-Threaded
Ha_runtime_up MSVC6.DLL	ha<VERSION>6s.dll	ha<VERSION>6t.dll
		SMP
Ha_runtime_smp MSVC6.DLL		ha<VERSION>6p.dll

<VERSION> = Current released version of Runtime, for example, 6.0.1.5.

Microsoft Windows injection note:

DLLs are stored in the System Directory. In order for multiple configurations to be simultaneously installed, the DLLs have different names from their corresponding lib files. You will need to know the DLL name only if you are injecting.

Injecting Runtime

1. Determine which application you wish to inject. You will need to know the filename of the application. Note: This may be different from the name of the shortcut to the application.
2. Choose the Runtime DLL you wish to inject. You will need to use the DLL that corresponds with the Runtime Product you are using. See "DLL Files" on page 25 for more information.
3. Open the injector control file for edit. Unless you have moved the injector control file, you should be able to edit this file via the shortcut "edit default gsInjector.txt" on the Runtime submenu of the Start menu.
4. Add a line to this file, providing the name of the application you wish to inject and the DLL file on one line, separated by a comma. For example:

```
MyExecutable.exe, ha60156s.dll
```

5. Make sure that the library can find a valid Runtime text license. See “The Geodesic Runtime Product License” on page 12 for more details.
6. Run the application as normal.



This injection technology will only work with GUI applications (strictly, those which rely on `User32.dll`). In particular, you may not be able to inject command-line applications.



There are two MSVC compiler options that will cause later injection to become impossible:

- The `/ML` or `/MLd` option causes MSVC to statically link in the single-threaded C-Runtime.
- The `/MT` or `/MTd` option causes MSVC to statically link in the multi-threaded C-Runtime.

If either of these has been used, Runtime cannot be injected.

How to check that your application has been successfully injected:

- If you have Microsoft Visual C++ installed, you can use the Process Viewer program. Start Process Viewer from the Visual C area of the Start Menu, then select the injected application and press the Memory Detail button. There should be a selector that lists all associated DLLs. This should include the Runtime injector DLL and the chosen Runtime library.
- Run the ModuleList utility, available from <http://www.microsoft.com/msj/0998/code/sep98hood.exe>

Moving the Injector File

To configure the injector, you will need to edit the Injector’s control file. This will normally be called `gsInjector.txt`, and by default will be placed in the Windows system directory (e.g. `c:\winnt\system32`). If you want Runtime to choose another file, set the environment variable `GS_INJECTOR_FILE`. User-set environment variables take precedence over the default location. (For information on setting environment variables, see the section “Setting Windows Environment Variables” below.) Because Runtime installs the shortcut “edit default `gsInjector.txt`” in the Runtime submenu of the Start menu, you may wish to edit this to point to the new injector control file location.

Uninstalling under Windows

If you no longer want to inject a particular application, then simply comment out (by preceding it with an asterisk symbol `(*)`) the relevant line in the injector control file (see the

section on Injector configuration). This change will take effect the next time the application is launched.

To uninstall Runtime completely, use the uninstall shortcut on the Runtime submenu of the Start menu, and then reboot.

Setting Windows Environment Variables

Environment variables may be set by the following methods:

- From a command window, use the set command. For example:

```
set GS_LICENSE_FILE=d:\remlic.txt
```



Setting the variable through the command window only affects those applications launched from that command window. Setting variables in the control panel affects all subsequently launched applications. Using `autoexec.bat` will cause the environment variable to set during the next login shell.

- Through the System applet in the Control Panel:

Open the control panel from the Settings area of the Start Menu. Double click on the System applet to launch it, and choose the Environment tab. You can type in the environment variable name/value pair in the text box.

Working with Services

To get Runtime to work with a service, you'll need to manually stop and start your application, the method of which can differ for each service.

For example, to work with IIS:

- 1) Stop Internet Information Services with: Start > Settings > Control Panel > Administrative Tools > Internet Services Manager > (right-click) *YourServerHere* > Restart IIS... > Stop Internet Services on *YourServerHere*.
- 2) Add the following line to `gsInjector.txt`:
`inetinfo.exe, ha60156t.dll`
- 3) Start Internet Information Services with: Start > Settings > Control Panel > Administrative Tools > Internet Services Manager > (right-click) *YourServerHere* > Restart IIS... > Start Internet Services on *YourServerHere*.

SUN SOLARIS

Runtime currently supports the following:

Architecture	Operating System	Compiler
Sun SPARC	Solaris 2.6	GNU 2.95.x +
Sun SPARC	Solaris 2.6	Visual Workshop 4.2
Sun SPARC	Solaris 2.6	Visual Workshop 5.0
Sun SPARC	Solaris 2.6	Forte 6.0
Sun SPARC	Solaris 7	GNU 2.95.x +
Sun SPARC	Solaris 7	Visual Workshop 4.2
Sun SPARC	Solaris 7	Visual Workshop 5.0
Sun SPARC	Solaris 7	Forte 6.0
Sun SPARC	Solaris 8	GNU 2.95.x +
Sun SPARC	Solaris 8	Visual Workshop 4.2
Sun SPARC	Solaris 8	Visual Workshop 5.0
Sun SPARC	Solaris 8	Forte 6.0

Installation

You must have root access for installation via `pkgadd`. “The Runtime Directories” on page describes the files that are installed.

Insert the Runtime CD-ROM and install using the command:

```
% pkgadd -d /cdrom <filename>
```



Product

<filename>

High Availability	GDSChaup
High Availability SMP	GDSChasmp

The Runtime Directories

The root directory depends upon the Solaris compiler and which Runtime product was installed.

Compiler	Product	Root Directory
GNU	High Availability	<UNPACKDIR>/ha_runtime_up /<VERSION>/sol/sparc/gnu
GNU	High Availability SMP	<UNPACKDIR>/ha_runtime_smp /<VERSION>/sol/sparc/gnu
VW4	High Availability	<UNPACKDIR>/ha_runtime_up /<VERSION>/sol/sparc/vw4
VW4	High Availability SMP	<UNPACKDIR>/ha_runtime_smp /<VERSION>/sol/sparc/vw4
VW5	High Availability	<UNPACKDIR>/ha_runtime_up /<VERSION>/sol/sparc/vw5
VW5	High Availability SMP	<UNPACKDIR>/ha_runtime_smp /<VERSION>/sol/sparc/vw5
Forte 6	High Availability	<UNPACKDIR>/ha_runtime_up /<VERSION>/sol/sparc/vw6
Forte 6	High Availability SMP	<UNPACKDIR>/ha_runtime_smp /<VERSION>/sol/sparc/vw6

<UNPACK-DIR> is the directory in which the unpack command for the tar file was issued. By default, pkgadd installs into /opt/.

<VERSION> is the current released version of Runtime, for example, 6.0.1.5.

The root directory will be referred to as <ROOT_DIR>. This directory contains:

- The readme, release notes, and installation files.
- A bin/ directory with the injector.
- A doc/ directory with the manual.
- An include/ directory containing C header files.
- A lib/ directory containing the libraries and object files.
- A man/ directory containing the man pages for gsinject.

Injecting Runtime

Under Sun Solaris, injection is part of the way an application is invoked. Whereas normally an application is invoked as a command with arguments, with injection the gsinject command must be used to launch the application as described below.

- 1) Choose an application to inject. You will need to know the command and arguments normally used to launch it.
- 2) Set the flag for the High Availability Runtime library.

- 3) Choose whether you want to inject a single- or multi-threaded library. It is always safe to use the multi-threaded or SMP library. It is only safe to use the single-threaded library if the application is single-threaded.
- 4) Decide whether child processes launched by the application should also be injected. If the child process is a different application than the parent, keep in mind that Runtime may not be compatible with the child. Forked processes are always injected. If a process is forked and then `execed`, it will only be injected if the `-e` flag is used.
- 5) Make sure that the library can find a valid Runtime text license (using either `GS_LICENSE_FILE` or the `-c` option). See “The Geodesic Runtime Product License” on page 12 for more details.
- 6) Run `gsinject` using the parameters described below. For example, to inject the High Availability Runtime multi-threaded library into `myprog` and all child processes, use the command:

```
gsinject -t -a -e myprog arg1 arg2
```

Syntax of the `gsinject` command

```
gsinject (-s|t|p) -a [-n|e] [-v] [-h] [-c license-file]
command [arguments]
```

Threading flags:

- s** Injects single-threaded flavor of the library.
- t** Injects multi-threaded flavor of the library.
- p** Injects SMP flavor of the library.

Library selection flags:

- a** Injects High Availability Runtime library.

Exec behavior flags:

- n** (Default) If the application launches child processes using `exec`, they will not be injected.
- e** Child processes launched via `exec` will be injected.

Other options:

- v** Verbose mode. Messages about injection will be emitted to standard output.
- h** Displays help text.
- c** Specifies the location of the text license file. Overrides the `GS_LICENSE_FILE` variable.



By default, Solaris uses lazy linking. Use `LD_BIND_NOW` to force non-lazy binding. See the troubleshooting item “I’m getting errors such as `ld.so.1: ld: fatal: relocation error: symbol not found:`” on page 56 for details.

Tuning Runtime's Performance

Out of the box, Runtime should already be configured to give you the best results. Even so, you can customize Runtime's performance in many ways. You'll find the most common methods discussed here.

“Definitions” on page 32

“When to Tune Runtime's Performance” on page 32

“When Not to Tune Runtime's Performance” on page 33

“How to Tune Runtime's Performance” on page 33

“Controlling When Garbage Collection Occurs” on page 34

“Tuning Runtime's Virtual Memory Performance” on page 35

“Optimization for Speed” on page 36

“Multi-Threading” on page 37

The techniques described here can help Runtime run faster or consume less space. However, these techniques are recommended only when under extreme performance pressure because Runtime is rarely the limiting factor in a program's performance.

DEFINITIONS

To *tune* or *optimize* a program means to increase its efficiency without otherwise changing its behavior. Optimization can increase a program's speed or reduce its space requirements.

WHEN TO TUNE RUNTIME'S PERFORMANCE

Any of the widely available performance-profiling tools can help determine where there is a performance bottleneck for your program. In the rare cases where it is necessary to tune Runtime's performance, the optimization techniques in this section can elevate Runtime's efficiency to meet the strictest performance requirements.

- If your program uses virtual memory, you can often tune Runtime to get dramatic performance improvements by following the procedure described above under the heading “Tuning Runtime's Virtual Memory Performance” on page 35. If your program runs too slowly *and* spends a large proportion of its time inside Runtime, tuning Runtime's performance can yield big dividends. However, programs rarely spend a large

proportion of their time inside Runtime, so you may want to use a profiling tool to confirm that this is the case.

- If your program consumes too much memory, you can tune Runtime to reduce its space consumption. Especially consider adjusting the values of `GS_PRIORITY` or setting `GS_DISABLE_FREE=0` (the default).

WHEN NOT TO TUNE RUNTIME'S PERFORMANCE

- If your program already has good performance, you don't need to optimize it. Optimizing Runtime's performance introduces extra complexity together with opportunity for error.
- If your program does not spend much of its time inside Runtime, there is little opportunity to improve its speed by tuning Runtime. The only relatively common situation where a program spends much of its time in Runtime is if it uses virtual memory, in which case you should consider following the procedure described under the heading "Tuning Runtime's Virtual Memory Performance" on page 35.

HOW TO TUNE RUNTIME'S PERFORMANCE

You can tune Runtime's performance by controlling when garbage collection occurs and by telling Runtime which memory to scan for pointers. There are also special opportunities for tuning virtual memory performance and multi-threaded programs.

CONTROLLING WHEN GARBAGE COLLECTION OCCURS

Identifying points where your program can spare time for Runtime to analyze its data structures can provide significant improvements in Runtime's performance. For example, if your program displays a series of results in the screen, the user will probably spend several seconds viewing them. Calling for a garbage collection immediately after displaying the results would allow Runtime to perform garbage collection when your program is most likely idle.



Tell Runtime how frequently to perform garbage collection cycles. You can instruct Runtime how frequently it should garbage collect by changing the value of `GS_PRIORITY`, the percentage of garbage allowed to accumulate between collections. The initial value of `GS_PRIORITY` is 40. Lower values increase collection frequency. At lower values, your programs will spend more time garbage collecting but will take up less space, as less memory is occupied by garbage. Higher values decrease collection frequency. Setting it to 10,000 (a factor of 100 times the live memory) or more will effectively stop Runtime from garbage collecting unless your program runs out of memory. Setting `GS_DISABLE_AUTOMATIC_GARBAGE_COLLECTION=1` will achieve the same result.

TUNING RUNTIME'S VIRTUAL MEMORY PERFORMANCE

Certain optimizations can substantially improve Runtime's virtual memory performance. *Virtual memory* is used by most operating systems to allow programs to run even when they do not fit in the computer's memory. The operating system copies the excess memory out to disk and only pages it back into physical memory when it is accessed.

Runtime determines what data is in use by tracing through all your program's data structures. If your program uses substantially more data than can fit in physical memory, Runtime will page in a lot of data from the disk during this analysis. Because disks are far slower than dynamic memory, this paging can have a major impact on Runtime's performance on extremely large programs. The optimizations described here dramatically improve Runtime's virtual memory performance.

Although the following technique allows Runtime to manage very large programs more efficiently by improving virtual memory performance, you should only attempt to optimize Runtime's virtual memory performance if your program thrashes the disk during collection.

Do not disable manual freeing of data. If you have called `GS_DISABLE_FREE=1`, `free()` and `delete` will not release memory (although `delete` always calls destructors). This protects you from premature frees but also makes your program use more memory. If your program uses large amounts of memory, you may experience better virtual memory performance and space utilization if you leave manual freeing enabled (the default).

OPTIMIZATION FOR SPEED

While Runtime takes various steps to ensure correct memory management in your program, there are still programming errors that may lead to run-time program corruption. These include:

- 1) Short memory overwrites (writing beyond the end of an object).
- 2) Reading from or writing to an object after its memory has been explicitly freed, a.k.a. premature free or use-after-free.
- 3) Explicitly freeing a memory block more than once, a.k.a. double frees or multiple frees.

Fast Mode

In fast mode, the allocation strategy is optimized for speed, relying on the correctness of your program. If your program isn't experiencing the problems listed above, and you want to experience Runtime's full performance benefits, you can turn off Robust Mode by setting the environment variable `GS_FAST_MODE=1`.

MULTI-THREADING

This section describes how to use Runtime to automatically manage memory in multi-threaded programs. Managing memory in multi-threaded programs is especially difficult because there is often no way to determine in advance which will be the last thread to use shared data. Runtime's automatic memory management eliminates this pitfall in managing shared data.

Definitions

Multi-threaded programs are programs with several simultaneous threads of execution. Threads are similar to processes except that all threads belonging to a program share the same address space. The operating system schedules these threads and will even run them in parallel if your computer has multiple processors, allowing programs to run many times faster. Multi-threading is supported by an increasing number of modern operating systems.

All code used in multi-threaded programs must be *thread-safe*, which means that it must be designed to work correctly even if it is being simultaneously called from multiple threads. Runtime includes thread-safe versions of the Runtime libraries for many operating systems that support multi-threading. Runtime also makes an SMP product that is thread-efficient.

When to Use Multi-Threading

Runtime's automatic memory management makes memory management easier in multi-threaded programs. You can still use Runtime to get all the usual benefits of automatic memory management under multi-threaded programs.

When Not to Use Multi-Threading

If your program is not multi-threaded, you don't need to use the thread-safe Runtime libraries. The thread-safe libraries are slightly slower than the single threaded Runtime libraries.

How to Use Multi-Threading

Use the thread-safe version of the Runtime libraries as per the platform specific information for your operating system. The thread-safe versions of the Runtime libraries are used normally.



To use Runtime with multi-threaded programs, you must use the thread-safe versions of the Runtime libraries. If you do not, your program will not work.

Choosing the Right Number of Heaps



SMP Product Only

Since there is extra memory overhead for the internal control structures and reserve memory of each heap, there is a trade-off between performance and memory use as more heaps are added. In general, when sufficient memory is available, it is desirable to use as many heaps as the maximum number of concurrent threads that will perform significant amounts of memory allocation, so that threads do not need to contend for the same heaps.

The optimal number of heaps will vary from application to application, and can be determined by iterative performance testing. On SMP machines, you should have at least as many heaps as processors, we recommend you have at least twice as many heaps (the default setting). Use the environment variable `GS_MULTI_HEAP` to set the number of heaps you wish to use.

Tuning Runtime's Reliability

Out of the box, Runtime should already be configured to give you the best results.

There are multiple ways to customize Runtime's reliability. The most common methods are discussed here.

“Optimization for Reliability” on page 39

“Automatically Fixing Premature Frees” on page 40

OPTIMIZATION FOR RELIABILITY

While Runtime takes various steps to ensure correct memory management in your program, there are still programming errors that may lead to run-time program corruption. These include:

- 1) Short memory overwrites (writing beyond the end of an object).
- 2) Reading from or writing to an object after its memory has been explicitly freed, a.k.a. premature free or use-after-free.
- 3) Explicitly freeing a memory block more than once, a.k.a. double frees or multiple frees.

Robust Mode

Robust Mode, the default mode of operation, addresses the concerns listed in the introduction by making the allocation strategy more robust against these errors. It allows writing to a freed object (assuming the object hasn't been reused or returned to the operating system through a footprint reduction) and performing multiple frees upon an object, without disastrous consequences to the program. In addition, robust mode allows you to add “extra padding” to allocated objects via `GS_ADD_BYTES`, if you feel that overwrites are happening, allowing your program to continue running. A direct result of this is that there will be slightly more memory use than using Fast mode (see page 36 details).

AUTOMATICALLY FIXING PREMATURE FREES

This section shows how to use Runtime to automatically fix premature frees.

A *premature free* occurs when a program frees memory that is still in use. It results from an incorrect call to `free()` or `delete`. Premature frees lead to program crashes, corrupt data, and incorrect program results. Premature frees are perhaps the most dangerous memory errors. Like other memory errors, they often occur intermittently. Even worse, incorrect results due to premature frees often look plausible, leading to damaging undetected errors.

Since Runtime eliminates memory leaks, by fixing premature frees Runtime can automatically fix all memory deallocation bugs in existing code.

Why Automatically Eliminate Premature Frees

Premature frees have serious consequences in programs written with manual memory management, resulting in crashes and corrupted data.

Premature frees are difficult to debug and often go undetected in programs. They are often intermittent. Even worse, memory allocators frequently reuse the same memory for the same type, resulting in plausible but incorrect values in prematurely freed objects. Without protection against premature frees, even the most reasonable-looking program results may be untrustworthy.

Third-party DLLs and libraries may have premature frees. Runtime can fix these even without rebuilding the DLLs and Libraries.

When Not to Automatically Eliminate Premature Frees

Programs use more memory when automatically eliminating premature frees because Runtime defers the release of memory until it verifies that the memory is unused. A small amount of memory may not be released at all if Runtime cannot be sure that it is safe to do so. This is not an issue for the vast majority of programs, but it may be significant in very large programs or programs with very strict memory requirements.

How Automatically Fixing Premature Frees Works

Setting the environment variable `GS_DISABLE_FREE=1` disables manual deallocation completely. This function tells Runtime to redefine `free()` and `delete` not to release memory, although `delete` still invokes destructors. Because `free()` and `delete` don't free memory, you are protected from prematurely releasing memory. Runtime will safely free the memory later when it is no longer in use.



Your program will use more memory when protection against premature frees is activated, since memory will not be released until Runtime determines it is safe. A small amount of memory may not be released at all if Runtime cannot ensure that it is safe. This is insignificant for the vast majority of programs, but it may be an issue in very large programs or programs with very strict memory requirements.

How To Automatically Fix Premature Frees

Set the environment variable `GS_DISABLE_FREE=1`.



While fixing premature frees, robust mode is pointless overhead except for memory overwrite protection. You can reduce this overhead by turning on fast mode with `GS_FAST_MODE=1`.

Administrator's Guide

CONFIGURING GEODESIC RUNTIME

This section explains how to configure Runtime at program initialization. There are two ways to configure Runtime:

- Configure Runtime with environment variables.
- Configure Runtime programmatically. See the programmer's reference.

When Not to Use Runtime Configuration

The default settings of Runtime are designed to be optimal for most programs; changing them may have subtle or unexpected consequences. Unless you are in an unusual situation, you may be better off letting Runtime make its own configuration decisions.

Configuring Runtime with Environment Variables

Many of Runtime's variables can be initialized with values taken from environment variables. Just define these variables in your environment before running the program. See "Geodesic Runtime Environment Variables" on page 43 for a list of environment variables that Runtime recognizes.

GEODESIC RUNTIME ENVIRONMENT VARIABLES

Please note that environment variables are equivalent in functionality to variables that the programmer can set on a line-by-line basis within the code. By default, environment variables are set to 0 or NULL. Setting the variable in the program code overrides the value of the environment variable. During injection, only the environment variables are available to be set.

Runtime libraries use some additional environment variables that begin with the string “GS_”, in addition to the ones listed below. To avoid unexpected behavior, do not set any environment variable that begins with “GS_” unless it is documented or has been explained by Geodesic technical support.



An environment variable being listed as  denotes that we recommend the user have in-depth knowledge of their program and the command before using it.

EnvironmentVariableName

Default: Default value if variable is not set.

Description: Description of what the environment variable does.

Restrictions: Restrictions for the use of the environment variable.

Related APIs: Corresponding function. This environment variable will be overridden by any setting of the corresponding function in your program. May also list other related APIs.

Comments and Warnings: Listing of any comments or warnings.

GS_ADD_BYTES

Default: 0

Description: GS_ADD_BYTES protects your application from code that writes outside allocated memory. When you set GS_ADD_BYTES, the size of any allocation request will be increased by the number of bytes specified, rounded up to the nearest word.

Restrictions: Works only in robust mode (default mode); will not work if GS_FAST_MODE is set or gsSetFastMode(GS_ON) has been called.

Related APIs: This environment variable will be overridden by any setting of gsSetAddBytes() in your program.

Comments and Warnings: When using robust mode, all allocation requests are already increased by two words (8 bytes on a 32-bit machine). Any settings of less than 8 bytes will therefore be ignored.

GS_ALLOW_USER_STACKS



Default: 0

Description: Sometimes programs implement their own user-level thread libraries that change the location of the stack. By default, Runtime is set to correctly handle this situation. Should you wish to not allow user stacks, you can set this environment variable to 0.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetAllowUserStacks()` in your program.

Comments and Warnings: The collection frequency may be too high if user stacks are used and `GS_ALLOW_USER_STACKS` is set to 0.

GS_COLLECT_AT_END

Default: 0

Description: By setting this to 1, Runtime will run an additional collection cycle at program exit. By default, Runtime will not collect at exit. See `gsSetCollectAtEnd()` for related information.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetCollectAtEnd()` in your program.

Comments and Warnings: None.

GS_DISABLE_AUTOMATIC_GARBAGE_COLLECTION



Default: 0

Description: This variable tells Runtime whether to automatically schedule garbage collection. If `GS_DISABLE_AUTOMATIC_GARBAGE_COLLECTION` is set, the garbage collector will not operate unless your program runs out of memory or you explicitly request garbage collection by calling `gsCollect()`.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetAutomaticGarbageCollection()` in your program.

Comments and Warnings: None.

GS_DISABLE_FOOTPRINT_REDUCTION

Default: 0

Description: This variable turns off both automatic and explicit heap footprint reduction. Runtime will therefore retain unused memory to fulfill future memory requests instead of returning memory to the operating system.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetFootPrintReduce()` in your program.

Comments and Warnings: If you wish to effectively disable automatic footprint reduction, set `GS_MEM_FREED_BEFORE_NEXT_FOOTPRINT_REDUCE` to 1000000000 (approximately 1 Gb). Manual footprint reduction will be available via `gsFootPrintReduce()`.

GS_DISABLE_FREE

Default: 0

Description: When `GS_DISABLE_FREE` is set, explicit calls to `free()` and `delete` will not immediately reclaim memory, as they do by default. Destructors will be promptly called when you delete an object, but the memory will be safely reclaimed after Runtime determines that it is safe. This protects you from prematurely releasing memory and makes reuse of existing code easier, but may make some programs use slightly more space. See “Automatically Fixing Premature Frees” on page 40.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetFixPrematureFrees()` in your program.

Comments and Warnings: If `realloc()` returns a new pointer, the memory of the old pointer will not be freed until Runtime ensures it is safe to do so.

GS_DO_NOT_INTERACT_WITH_DESKTOP



Default: 0

Description: This integer variable stops Runtime from displaying errors in a dialog box. The default 0 (zero) setting directs Runtime to use dialog boxes to display error

messages. When this variable is set to 1, the errors are written to the log file. This variable is used when development is done using an NT service or a console application that cannot interact with the desktop.

Restrictions: Microsoft Windows Only

Related APIs: This environment variable will be overridden by any setting of `gsSetDesktopInteraction()` in your program.

Comments and Warnings: None.

GS_DONT_LOG_WITH_PID



Default: 0

Description: By default, every instance of the program will write to a unique log file name containing its process ID, and will not overwrite earlier log files written by the same executable. If `GS_DONT_LOG_WITH_PID` is set to 1, the leaks will be written to `gs.log` in the directory specified by `GS_LOG_FILE_PATH` and will overwrite the previous `gs.log` produced by the same executable.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetLogWithPID()` in your program.

Comments and Warnings: Turning PID logging off will not get rid of the PID from all log files. If your process forks off other processes, each of these will have their own log file. To be able to distinguish among log files, all these child log file names will contain the PID. On Windows, log file names for injected processes always contain the PID.

By default, for performance reasons, Runtime does not report memory errors and leaks it finds, which means this variable normally has no effect. However, if the function `gsSetReportLeaksSwitch()` is set to `GS_ON` (or the `GS_REPORT_LEAKS` environment variable is set), Runtime logs errors and leaks to an external file (which slows performance). If `gsSetPrintStats()` is set to `GS_ON`, or the environment variable `GS_PRINT_STATS` is set, the resulting log file will be modified by `GS_DONT_LOG_WITH_PID`.

GS_FAST_MODE

Default: 0

Description: Setting this to 1 enables fast mode, optimizing the allocation strategy for speed. See “Optimization for Speed” on page 36 for information regarding fast mode.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetFastMode()` in your program.

Comments and Warnings:

GS_IGNORE_OFF_PAGE_POINTERS

Default: 0

Description: If this is set to 0, Runtime will respect pointers to the second and subsequent pages of an object for the purposes of determining if an object is retained. If this is set to 1, Runtime will consider a very large object as leaked if there is no pointer into its first page. This can help identify very large leaked objects even if there is a stray pointer into them. `malloc()` and `new` will use `gsMallocIgnoreOffPage()` to allocate objects larger than 65,535 bytes. You can change the threshold for very large objects by redefining `gsSetVeryLargeAllocationSize()`. See “Ignoring Pointers Beyond the First Page” in the programmer’s reference for usage information.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetIgnoreOffPagePointers()` in your program.

Comments and Warnings: None.

GS_INJECTOR_FILE

Default: `<system_directory>\gsInjector.txt`

Description: To configure the injector, you will need to edit the Injector’s control file. This will normally be called `gsInjector.txt`, and by default will be placed in the Windows system directory (e.g. `c:\windows\system32`). If you want Runtime to choose another file, set the environment variable `GS_INJECTOR_FILE`. User-set environment variables take precedence over the default location. (For information on setting environment variables, see the section “Setting Windows Environment Variables” on page 27.) Because Runtime installs the shortcut “edit default `gsInjector.txt`” in the Runtime submenu of the Start menu, you may wish to edit this to point to the new injector control file location.

Restrictions: Microsoft Windows Only

Related APIs: None.

Comments and Warnings: None.

GS_LICENSE_FILE

Default: 0

Description: This variable is used to register the name and location of the text license file. See “Text Licenses” on page 12 for details.

Restrictions: None.

Related APIs: None.

Comments and Warnings:



If you do not set this variable, Runtime will look for `gslicense.txt` in the system directory (e.g. `c:\winnt\system32`). For information on setting environment variables, see the section “Setting Windows Environment Variables” on page 109.



On Unix, except for HP-UX, when using injection, the `-c` option can also be used to specify the location of the text license file. See the appropriate appendix for your operating system for more information.

GS_LOG_ALL_LEAKS



Default: 0

Description: This environment variable declares an integer that specifies if all leaks are to be logged to the log file. If set to 1, all leaks will be written to the defined log file (`gs.log` by default) so long as `GS_REPORT_LEAKS` is set or you have called `gsSetReportLeaksSwitch(GS_ON)`. This overrides the default setting of 500 leaks reported to the log file.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetLogAllLeaks()` in your program.

Comments and Warnings: None.

GS_LOG_FILE_NAME

Default: `gs`

Description: This variable is the name of the file to which Runtime prints statistics, warnings, errors, and leak reports.

Restrictions: No logging will take place unless `gsSetReportLeaksSwitch()` or `gsSetPrintStats()` is set to `GS_ON` or the environment variable `GS_REPORT_LEAKS` or `GS_PRINT_STATS` is set.

Related APIs: This environment variable will be overridden by any setting of `gsSetLogFileName()` in your program.

Comments and Warnings: `.log` is automatically appended.

GS_LOG_FILE_PATH

Default: unset, thus creating a log file in the temp directory.

Description: Sets a path (and drive, on Microsoft Windows file systems) for the log file, if any.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetLogFilePath()` in your program.

Comments and Warnings: On UNIX, defaults to `/tmp`. On Windows, defaults to the value of `$TEMP` or `$TMP`.

GS_LOG_WITH_HOSTNAME

Default: 0

Description: If `GS_LOG_WITH_HOSTNAME` is set, the host name is added to the log file as it appears in `HOSTNAME` environment variable. This will vary from system to system. On some systems, `HOSTNAME` is set to the shortened host name (example: “duke”). On other systems, `HOSTNAME` is set to the full host name (example: “duke.company.com”).

Restrictions: UNIX only

Related APIs: This environment variable will be overridden by any setting of `gsSetLogWithHostname()` in your program.

Comments and Warnings: None.

GS_MEM_FREED_BEFORE_NEXT_FOOTPRINT_REDUCE

Default: 1024000 bytes (1 MB)

Description: Runtime periodically returns memory to the operating system. The default regularity is 1024000 bytes (1 MB) of freed memory. You can change the frequency of footprint reduction by altering this value in bytes. This can be useful if your application has no other applications running on the same system.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetMaxMemFreedForFootPrintReduce()` in your program.

Comments and Warnings: If you wish to effectively prevent automatic footprint reduction, set this to 1000000000 (approximately 1 Gb); manual footprint reduction will still be available via `gsFootPrintReduce()`.

GS_MULTI_HEAP

Default: unset, defaults to 2x the number of processors

Description: This variable sets the number of heaps used internally for allocation, to avoid thread contention.

Restrictions: SMP Product Only

Related APIs: This environment variable will be overridden by any setting of `gsSetNumberOfHeaps()` in your program.

Comments and Warnings: See “Choosing the Right Number of Heaps” on page 38 for more information.

GS_PRINT_STATS

Default: 0

Description: Setting this variable to 1 tells Runtime to log statistics on its operation. This will cause Runtime to periodically print statistics on its behavior and memory usage to the log file.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetPrintStats()` in your program.

Comments and Warnings: None.

GS_PRIORITY

Default: 40

Description: This variable supplies an initial integer value for `gsSetGCPriority()`. Larger values decrease the frequency of automatic garbage collection, and a value of 10,000 (100 times the live memory) or more will effectively prevent it, a result more reliably achieved by defining `GS_DISABLE_AUTOMATIC_GARBAGE_COLLECTION` or setting `gsSetAutomaticGarbageCollection(GS_OFF)`.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetGCPriority()` in your program.

Comments and Warnings: None.

GS_REPORT_LEAKS

Default: 0

Description: When this is set to 1, Runtime will create a log file with brief reports on memory leaks and other memory management errors.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetReportLeaksSwitch()` in your program.

Comments and Warnings: None.

GS_SET_SCAN_ALIGNMENT

Default: The default value depends on the compiler.

Description: This variable tells Runtime how to scan for pointers. For example, setting the scan alignment to 4 says that pointers are all stored on 4 byte boundaries, although they may point to any address (aligned or not). If you pack your data structures, you may need to change this value accordingly.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetScanAlignment()` in your program.

Comments and Warnings: Setting this to anything but word boundaries will significantly slow down garbage collection.

GS_STOP_SIGNAL



Default: 0

Description: Runtime uses signals to put threads to sleep during a garbage collection cycle. Runtime searches for an available, unused signal at runtime. If you feel you need to, you can manually set the signal with `GS_STOP_SIGNAL`.

Restrictions: HP-UX, IBM AIX, and Linux (multi-threaded and SMP) Only

Related APIs: This environment variable will be overridden by any setting of `gsSetStopSignal()` in your program

Comments and Warnings: Be very careful with changing this as it can lead to thread problems.

GS_ZERO_ALLOCATED_OBJECT

Default: 0

Description: When set to 1, this variable tells Runtime to zero non-leaf objects.

Restrictions: None.

Related APIs: This environment variable will be overridden by any setting of `gsSetZeroAllocatedObject()` in your program.

Comments and Warnings: None.

DEPRECATION INFORMATION

Throughout the development of Runtime, certain functions have been changed or removed. If you were using a previous version of Runtime, the following tables may help you relate the functions and variables you used before with the new functions and variables.

Log File

REMIDI¹ V5.x	Runtime V6.0
gs.log	gs-<PID>.log

Environment Variables

REMIDI¹ V5.x	Runtime V6.0
GS_ADD_PROCESS_ID_TO_LOG_FILE_NAME	GS_DONT_LOG_WITH_PID
GS_COLLECT_HEAP_ALLOC	Removed
GS_HEAP_ALLOC	Removed
GS_IGNORE_OFF_PAGE	GS_IGNORE_OFF_PAGE_POINTERS
GS_LOG_FILE	GS_LOG_FILE_NAME

¹ REMIDI is a product of Geodesic Systems, Inc., which is not affiliated in any way with Remedy Corporation.

GEODESIC RUNTIME LOG FILE

Runtime optionally produces an ascii log of its activity. By default, Runtime does not generate a log file; you can activate logging with the environment variable `GS_REPORT_LEAKS`. The log file provides the following information:

- Memory leaked by your program. It provides the object's address and approximate size.
- Other memory errors.
- Statistics produced by `GS_PRINT_STATS=1`.
- Internal error messages from Runtime, which may be of use to Geodesic technical support.

How a log file name is constructed:

```
<Path>\<File Name>[-<Process ID>][-<Host Name>].log
```

- The default log file name is `gs`. You can change the name of the log file with `GS_LOG_FILE_NAME`.
- `.log` is automatically appended.
- Logging with the Process ID is turned on by default, making the full default log name `gs-<Process ID>.log`.
- The location for the log file defaults to the temp directory.
 - Unix: `"/tmp"`
 - Windows: the value of the `$TEMP` or `$TMP` environment variables.
 - You can set the path and, on Windows, the drive using `GS_LOG_FILE_PATH`.

Examples:

A log file using the defaults = `\tmp\gs-12345.log`

A log file with the path set to the local directory, the name set to "myApp", and host name adding turned on (under UNIX) = `./myApp-12345-duke.log`

The number of leaks reported to the log file is limited to 500. To override this value and report *all* leaks, set the environment variable `GS_LOG_ALL_LEAKS`.



Using just `GS_PRINT_STATS=1` will activate the log file, however, no leak information will be reported. You will only get information on Runtime's behavior and memory usage.

Troubleshooting Guide

This section addresses problems or issues you may run into. If your problem is not described here, please contact Geodesic Systems, Inc. Technical Support, and we'll work with you personally.

TROUBLESHOOTING INJECTING WITH GEODESIC RUNTIME

"My application crashed upon injection."

You may have injected a Runtime library that was designed for a different compiler. It is necessary to inject the library associated with the same compiler that was originally used to compile the application. For example, on Microsoft Windows, if you inject an application with an MSVC6 library and it crashes, this may be because the application was compiled with VisualAge.

"When I start my injected application, I got the following error: 'This unlicensed Geodesic Systems product may only be used with the sample programs...'"

This probably means that the library was unable to find a valid license. Make sure that the text license file is available, is not corrupted, and is pointed at by the `GS_LICENSE_FILE` environment variable.

If this still does not correct your problem, verify that the license has not expired, is for the correct product, application (if application specific), and platform.



On Microsoft Windows, file extensions can be hidden from view. Verify that the file has not been accidentally named `gslicense.txt.txt` or something similar.

TROUBLESHOOTING RUNNING WITH GEODESIC RUNTIME

"Runtime is not returning as much memory to the OS as I would like."

You can address this by setting the environment variable `GS_PRIORITY` to a lower value. Also verify that and the environment variable `GS_DISABLE_FOOTPRINT_REDUCTION` is set to 0.

"The log file says 'gcGetCleanMemory: MMAP uncommitted failed'."

This happens when Runtime tries to allocate more virtual memory but the virtual memory system is unable to accommodate the request. Increase the amount of available virtual memory or swap space to correct for this.

"My application is hanging or deadlocking."

A "hang" occurs when a program reaches an infinite state with no possibility of exiting. One example of a hang that may occur in a multi-threaded environment is a "deadlock". A deadlock occurs when two threads are blocked on each other waiting for a resource that the other has acquired.



The use of `fork` with multi-threaded products is discouraged. Geodesic products have no way of guaranteeing that child processes will run correctly if multiple threads have been created by the parent process.

How to check to see if a program is hung or deadlocked:

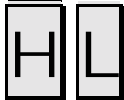


Execute the `top` command. This will display CPU and memory use for each process. If your process is not using much CPU then it may be hung.

Possible fixes:



AIX only: Make sure the `AIXTHREAD_SCOPE` environment variable is set to `S` (system-wide). This is a requirement for running Runtime in a multithreaded environment.



Runtime searches for an available, unused signal at runtime. If you feel this may be the problem, try setting the environment variable `GS_STOP_SIGNAL` to a signal that is not in use. For more information, see `GS_STOP_SIGNAL` in "Geodesic Runtime Environment Variables" starting on page 43.

"I'm getting errors such as `ld.so.1: ld: fatal: relocation error: symbol not found:`"



Sun Solaris only.

By default, Solaris uses lazy linking. This means that the dependencies for functions from shared libraries are resolved the first time the function is called. Non-lazy linking forces the dependencies to be resolved when the library is first loaded. `LD_BIND_NOW` is a Solaris standard environment variable that can be set to force non-lazy binding. Alternatively, the `-z now` flag may be specified in the link line to require non-lazy binding.

Using `LD_BIND_NOW` does not actually solve this problem. The real problem is missing definitions for functions. With `LD_BIND_NOW`, errors of this type will be seen immediately upon execution of the program. Without this, the errors may not be seen until much later and may make it all the way to the distributed software.

Appendix

INTRODUCTION TO MEMORY MANAGEMENT

In order to better understand how Geodesic Runtime will be useful to you and your organization, following is a brief discussion of memory management and how it can affect the performance and reliability of your application.

What is Memory Management?

Memory management is the process of coordinating and controlling the use of memory in a computer system. Memory management has three major areas: 1) memory management hardware, 2) operating system memory management, and 3) application memory management. The most commonly accessible, and therefore the most error-prone, area of memory management is application memory management. Application memory management deals with allocation, deallocation, and garbage collection. Reliability and performance are affected most severely by application memory management errors.

Memory management software can help alleviate coding problems related to manually managing data objects; this automatic memory management is placed into one module. This module or subsystem keeps track of the memory storage and manages it in a standard, efficient manner instead of having it managed in a potentially error-prone, case-by-case basis inside the application.

Common problems in memory management and their effects

Reliability

Reliability is crucial in any application no matter how large or small. Unfortunately, the reliability of most applications can be severely damaged by common application memory management issues: memory leaks, memory overwrites, and premature frees.

Memory leaks

Memory is considered leaked when a data object is created but then not released when the program is done using it. Over time this leads to a build-up of unused memory, wasting space, causing program slow downs, and often causing program crashes. This is the most common problem caused by managing memory inside an application itself.

Memory Overwrites

Sometimes errors in logic or oddities in input can result in programs writing beyond the end of an object. This results in data corruption, incorrect information, and program crashes.

Premature frees

The opposite problem from memory leaks, premature frees are when memory is freed before the application is done working with it. This usually causes incorrect results when the application later tries to access this memory and gets whatever data now happens to be in that space.

Performance

The performance of an application is often what sets it apart from other applications. If one application runs faster or uses less memory than the rest, it will have an advantage over its competition. As with reliability, some common application memory management issues can dramatically affect the performance of an application: fragmentation, excessive memory usage, and heap contention.

Fragmentation

As memory objects of various sizes are allocated and released, the actual physical memory becomes a patchwork of used and unused spaces. This wastes time and memory, as the application has to go to greater lengths to find space for new allocations.

Excessive memory use

Sometimes memory management issues can keep an application from reaching peak performance. Excessive memory use is one such example. As an application's memory needs grow and shrink with load, it keeps as much memory for itself as the maximum amount it ever needed, even if most of that memory is not in use at the time.

Heap contention

Just as excessive memory usage affects the operating system's memory management, heap contention can counteract advances in memory management hardware. Standard allocators usually keep one memory heap, even for programs with multiple threads of execution. Any thread that allocates or deallocates data must lock the heap. Therefore, even though the threads can take advantage of multiple processors to speed up execution, they must still wait to allocate memory one at a time.

Under a single heap allocator, dynamically allocated memory from multiple threads may share a page in the heap. Therefore on an SMP system, if such threads get bound to different processors, processors will waste time as they refresh pages in cache that other threads have modified.

Solutions to common problems in memory management

Reliability

Garbage Collection

A solution that a memory management subsystem can offer to minimize memory leaks is to keep track of the memory itself and periodically “garbage collect” it. This means checking for memory that is no longer in use and freeing it back to the application’s heap.

Object Padding

Memory overwrites can be solved by increasing the size of objects beyond the normal length; this is typically by a few bytes. A memory management subsystem can also add an end tag to an object to detect writes beyond the end of the object.

Disabling Manual Deallocation

To address the problem of premature frees, a memory management subsystem can disable application deallocation calls and enable garbage collection. Then memory is not deallocated at the time of request, but rather when the memory is determined to be no longer in use. Premature and other “wild” frees become impossible.

Performance

Size-specific allocation

By organizing objects into different sections of memory based on their size, a memory management subsystem can reduce memory fragmentation and free list scan times. This is sometimes referred to as “segregated allocation.”

Footprint reduction

If a significant amount of memory has been freed back to the application’s heap, a memory management subsystem can release it back to the operating system for use by other processes.

Multiple heaps

To accelerate the execution of threads, a memory management subsystem can keep separate memory heaps for each one. Threads then no longer need to wait for each other to finish memory tasks and cache refreshes among processors is reduced. This can bring enormous performance gains in multi-threaded systems.

Geodesic Runtime as your solution

Through the use of state-of-the-art algorithms and technology, Geodesic Runtime protects against the most common memory management errors. This includes protecting the reliability and enhancing the performance of an application. Which Geodesic Runtime

product is right for you will depend upon your system requirements and the problems you are experiencing. See “The Geodesic Product Family” on page 61 for a description of the different versions of Geodesic Runtime.

THE GEODESIC PRODUCT FAMILY

High-performing, reliable software is essential for any company that wants to succeed. Whether you're developing software in-house or running large third-party applications, you need your software to work as well as possible. You're at the greatest risk if you rely on servers or environments where every minute of downtime means lost customers or lost revenue. Whether you produce such mission-critical applications or merely depend on them, Geodesic can reduce your risk.

Geodesic has two families of products to ensure application performance, availability and reliability: Geodesic Runtime and Great Circle®. Geodesic Runtime comes in four flavors.

Geodesic Runtime

Run-time Solution for Deployed Applications

Geodesic Runtime is a set of software libraries that can inject into your application when source code and object libraries are unavailable or link into your application before deployment. It integrates easily into both legacy applications and software in development. Because installation doesn't require any change to your source code or applications, it won't take long to get started.

Geodesic Runtime brings speed and reliability to under-performing applications; it can instantly turn a hopelessly unstable application into a bulletproof system. Add Geodesic Runtime to any C or C++ application and it will run faster, more reliably and with lower memory requirements, all in real time.

- *Geodesic High Performance Runtime* provides performance improvements optimized for a uniprocessor environment.
- *Geodesic High Performance Runtime SMP* provides performance improvements optimized for an SMP environment.
- *Geodesic High Availability Runtime* provides runtime diagnosis and remediation optimized for a uniprocessor environment.
- *Geodesic High Availability Runtime SMP* provides runtime diagnosis and remediation optimized for an SMP environment.

	Uniprocessor	SMP
Performance Optimized	Geodesic High Performance Runtime	Geodesic High Performance Runtime SMP
Reliability Optimized	Geodesic High Availability Runtime	Geodesic High Availability Runtime SMP

Great Circle

Real-time Error Detection and Code Diagnosis for Developers

Great Circle is an advanced diagnostic environment that allows developers to test, diagnose, and resolve memory problems. It pinpoints elusive errors that can cause excessive application memory use and devastating crashes, and its Web browser interface allows remote debugging through a LAN or over the Internet. Use Great Circle to validate software and reduce the risk of application and system failure.

Glossary

These definitions relate to how Runtime works and are not just general meanings.

Term	Definition
Appropriate time	Runtime releases memory when it is safe to do so, i.e. when the program has finished using it.
Blocking on Locks	In a multiple-thread environment, a thread making a request of the allocator must wait until any prior thread already using the allocator has finished.
Cache	A portion of memory existing on the processor chip, allowing faster data access than going to main memory (RAM). There may be multiple "levels" of cache.
Container Library	A library that supports a number of different ways to store data: sets, multi-sets, lists, hash tables, stacks, etc.
Destructor	Procedure that runs when an object is deleted. Similar to a finalizer.
Footprint	Refers to the amount of RAM any given program uses when loaded.
Finalizer	Procedure that runs when a garbage collector reclaims an object. Similar to a destructor.
Fragmentation	When memory becomes a patchwork of used and unused spaces as memory objects of different sizes are allocated and released.
GUI Libraries	Class libraries that aid in the development of Graphical User Interface applications.
Heap Contention	Multiple threads attempting to use the same allocator; leads to threads having to wait to allocate or release memory one at a time.
Macro	A definition that the compiler or preprocessor uses to expand different programming constructs when it processes a source file.
Memory Leak	Occurs when a data object has been allocated but not released after the program has finished using it.
Multi-Threaded programs	Programs with several simultaneous threads of execution within the same process.

Term	Definition
Non-aligned Data Structure	Object that starts at a non-word boundary.
Premature Free	Occurs when a data object is freed before the program has finished using it. Results from an incorrect call to free() or delete.
Reference Counting	A method of keeping track of data use by adding a counter to each element of how many pointers are pointing to it.
Run-time Diagnosis and Remediation	Automatic detection, diagnosis, and resolution of memory management problems.
SMP	Symmetric Multi-Processing. Using multiple processors that share the same physical memory in the same computer.
Thread-Efficient	Guarantees that multiple callers get safe heap access and they can execute significant portions of the routine in parallel on separate processors.
Thread Contention	Multiple threads attempting to use the same allocator; leads to threads having to wait to allocate or release memory one at a time.
Thread-Safe	Guarantees that multiple threads may safely call the allocator concurrently without corrupting the heap. Does not guarantee that the routine's code will be executed in parallel.
Value Semantics	Libraries that use value semantics store copies of the elements passed to them, rather than storing pointers to the elements.
Weak Pointer	A pointer that is not meant to be traced by the collector.
Word	A unit of memory, e.g. four (4) bytes on a 32-bit system.
Zeroed Memory	Blocks of memory allocated with all bytes set to zero.

Notices

PROPRIETARY RIGHTS

Portions of the Geodesic software include modifications to code which were released publicly by Xerox Corporation, subject to the requirement that the following notice be retained and included in the modified code: Copyright 1988, 1989 Hans-J. Boehm, Alan J. Demers. Copyright (C) 1991-1995 by Xerox Corporation. All rights reserved. THIS MATERIAL IS PROVIDED, AS IS, WITH ABSOLUTELY NO WARRANTY EXPRESS OR IMPLIED. ANY USE IS AT YOUR OWN RISK.

Copyright (c) 2001, Geodesic Systems, Inc. All rights reserved.

Index

C

compilers	
supported.....	7
configuration.....	42
customization	
performance.....	32
reliability.....	39

D

deprecation.....	52
dialog box.....	<i>See</i>
GS_DO_NOT_INTERACT_WITH_DESKTOP	
directories.....	10
HP-UX.....	14
IBM AIX.....	17
Linux.....	21
Microsoft Windows.....	24
Sun Solaris.....	28
DLL	
Microsoft Windows.....	25

E

environment variables.....	43
configuring Runtime.....	42

F

fast mode.....	36
footprint reduction	
defined.....	59
disable.....	45
tuning.....	49

G

garbage collection	
explained.....	59
scheduling.....	34
Geodesic	
contacting.....	3
product family.....	61
glossary.....	63
gs.log.....	48
GS_ADD_BYTES.....	43
GS_ALLOW_USER_STACKS.....	44
GS_COLLECT_AT_END.....	44
GS_DISABLE_AUTOMATIC_GARBA	
GE_COLLECTION.....	44
GS_DISABLE_FOOTPRINT_REDUC	
TION.....	45
GS_DISABLE_FREE.....	45
GS_DO_NOT_INTERACT_WITH_DE	
SKTOP.....	45
GS_DONT_LOG_WITH_PID.....	46
GS_FAST_MODE.....	46
GS_IGNORE_OFF_PAGE_POINTER	
S.....	47
GS_INJECTOR_FILE.....	47
GS_LICENSE_FILE.....	47
GS_LOG_ALL_LEAKS.....	48
GS_LOG_FILE_NAME.....	48
GS_LOG_FILE_PATH.....	49
GS_LOG_WITH_HOSTNAME.....	49
GS_MEM_FREED_BEFORE_NEXT_	
FOOTPRINT_REDUCE.....	49
GS_MULTI_HEAP.....	49
GS_PRINT_STATS.....	50
GS_PRIORITY.....	50
GS_REPORT_LEAKS.....	50
GS_SET_SCAN_ALIGNMENT.....	51

GS_STOP_SIGNAL..... 51
 GS_ZERO_ALLOCATED_OBJECT .51

H

heaps
 selecting the right number..... 38

I

icons 5
 injection 11
 HP-UX..... 15
 IBM AIX..... 18
 Linux..... 22
 Microsoft Windows..... 25
 Sun Solaris 29
 troubleshooting..... 54
 installation 9
 HP-UX..... 14
 IBM AIX..... 17
 Linux..... 21
 Microsoft Windows..... 24
 Sun Solaris 28

L

license
 text 12
 log file..... 53
 changing name 48
 hostname..... 49
 setting path 49

M

memory management
 introduction..... 57
 multi-threading 37
 choosing number of heaps..... 38

N

notices 65

O

obsolete52
 operating systems
 supported7

P

performance
 tuning.....32
 platforms
 supported7
 pop-up dialog box *See*
 GS_DO_NOT_INTERACT_WITH_
 DESKTOP
 premature frees
 automatically fixing..... 40, 45
 process ID *See*
 GS_DONT_LOG_WITH_PID
 proprietary rights65

R

read after free .. *See* GS_DISABLE_FREE
 reliability
 tuning.....39
 robust mode39
 running
 troubleshooting55

S

services
 working with.....27
 signals *See* GS_STOP_SIGNAL
 speed
 tuning.....36
 system requirements.....7

T

technical support.....3
 thread library
 user .*See* GS_ALLOW_USER_STACKS
 thread-safety37
 troubleshooting
 injection54

running.....	55
tuning.....	32
reliability	39
speed	36
typeface.....	5

U

use after free *See* GS_DISABLE_FREE

V

virtual memory	
tuning performance.....	35

W

write after free..*See* GS_DISABLE_FREE