Red Hat Developer Toolset 4.1

User Guide

Installing and Using Red Hat Developer Toolset

Last Updated: 2017-10-11

Red Hat Developer Toolset 4.1 User Guide

Installing and Using Red Hat Developer Toolset

Jaromír Hradílek
Red Hat Customer Content Services

Matt Newsome
Red Hat Software Engineering

Robert Krátký
Red Hat Customer Content Services
rkratky@redhat.com

Abstract

Red Hat Developer Toolset is a Red Hat offering for developers on the Red Hat Enterprise Linux platform. The Red Hat Developer Toolset User Guide provides an overview of this product, explains how to invoke and use the Red Hat Developer Toolset versions of the tools, and links to resources with more in-depth information.

Table of Contents

PART I. INTRODUCTION

CHAPTER 1. RED HAT DEVELOPER TOOLSET

1.1. ABOUT RED HAT DEVELOPER TOOLSET

Red Hat Developer Toolset is a Red Hat offering for developers on the Red Hat Enterprise Linux platform. It provides a complete set of development and performance analysis tools that can be installed and used on multiple versions of Red Hat Enterprise Linux. Executables built with the Red Hat Developer Toolset toolchain can then also be deployed and run on multiple versions of Red Hat Enterprise Linux. For detailed compatibility information, see Section 1.3, “Compatibility”.

Red Hat Developer Toolset does not replace the default system tools provided with Red Hat Enterprise Linux 6 or 7 when installed on those platforms. Instead, a parallel set of developer tools provides an alternative, newer version of those tools for optional use by developers. The default compiler and debugger, for example, remain those provided by the base Red Hat Enterprise Linux system.

What Is New in Red Hat Developer Toolset 4.1

Red Hat Developer Toolset 4.1 provides a new docker-formatted container image: rhscl/devtoolset-4-perftools-rhel7, which includes components for monitoring system performance and optimizing code. See Section 1.8, “Using Red Hat Developer Toolset Container Images” for more detailed information about Red Hat Developer Toolset container images and instructions on how to use them. See Table 1.1, “Red Hat Developer Toolset Components” for a complete list of components included in Red Hat Developer Toolset 4.1.

Since Red Hat Developer Toolset 4.1, the Red Hat Developer Toolset content is also available in the ISO format at https://access.redhat.com/downloads, specifically for Server and Workstation. Note that packages that require the *Optional* channel, which are discussed in Section 1.5.3, “Installing Optional Packages”, cannot be installed from the ISO image.

Table 1.1. Red Hat Developer Toolset Components

Name	Version	Description
Eclipse	4.5.2	An integrated development environment with a graphical user interface.[a]
GCC	5.3.1	A portable compiler suite with support for C, C++, and Fortran.
binutils	2.25.1	A collection of binary tools and other utilities to inspect and manipulate object files and binaries.
elfutils	0.166	A collection of binary tools and other utilities to inspect and manipulate ELF files.
dwz	0.12	A tool to optimize DWARF debugging information contained in ELF shared libraries and ELF executables for size.
GDB	7.11	A command line debugger for programs written in C, C++, and Fortran.
ltrace	0.7.91	A debugging tool to display calls to dynamic libraries that a program makes. It can also monitor system calls executed by programs.

Name	Version	Description
strace	4.10	A debugging tool to monitor system calls that a program uses and signals it receives.
memstomp	0.1.5	A debugging tool to identify calls to library functions with overlapping memory regions that are not allowed by various standards.
SystemTap	2.9	A tracing and probing tool to monitor the activities of the entire system without the need to instrument, recompile, install, and reboot.
Valgrind	3.11.0	An instrumentation framework and a number of tools to profile applications in order to detect memory errors, identify memory management problems, and report any use of improper arguments in system calls.
OProfile	1.1.0	A system-wide profiler that uses the performance monitoring hardware on the processor to retrieve information about the kernel and executables on the system.
Dyninst	9.1.0	A library for instrumenting and working with user-space executables during their execution.

[a] If you intend to develop applications for Red Hat JBoss Middleware or require support for OpenShift Tools, it is recommended that you use Red Hat JBoss Developer Studio.

Red Hat Developer Toolset differs from "Technology Preview" compiler releases previously supplied in Red Hat Enterprise Linux in two important respects:

1. Red Hat Developer Toolset can be used on multiple major and minor releases of Red Hat Enterprise Linux, as detailed in Section 1.3, "Compatibility".

2. Unlike Technology Preview compilers and other tools shipped in earlier Red Hat Enterprise Linux, Red Hat Developer Toolset is fully supported under Red Hat Enterprise Linux Subscription Level Agreements, is functionally complete, and is intended for production use.

Important bug fixes and security errata are issued to Red Hat Developer Toolset subscribers in a similar manner to Red Hat Enterprise Linux for two years from the release of each major version release. A new major version of Red Hat Developer Toolset is released annually, providing significant updates for existing components and adding major new components. A single minor release, issued six months after each new major version release, provides a smaller update of bug fixes, security errata, and new minor components.

Additionally, the Red Hat Enterprise Linux Application Compatibility Specification also applies to Red Hat Developer Toolset (subject to some constraints on the use of newer C++11 language features, detailed in Section 3.2.4, "C++ Compatibility").

IMPORTANT

Applications and libraries provided by Red Hat Developer Toolset do not replace the Red Hat Enterprise Linux system versions, nor are they used in preference to the system versions. Using a framework called **Software Collections**, an additional set of developer tools is installed into the `/opt/` directory and is explicitly enabled by the user on demand using the `scl` utility.

1.2. MAIN FEATURES

The Red Hat Developer Toolset version of the **GNU Compiler Collection** (**GCC**) has been upgraded to version 5.3.1 with many bug fixes.

The version of the **GNU Debugger** (**GDB**) included in Red Hat Developer Toolset provides the new features, including the following:

- Support for per-inferior thread numbers has been added.
- **GDB** now allows users to specify explicit locations, bypassing the linespec parser. This feature is also available to GDB/MI clients.
- **GDB** now has support for fork and exec events on remote-mode Linux targets. This enables `follow-fork-mode`, `detach-on-fork`, and `follow-exec-mode` modes and fork and exec catchpoints.

Additionally, the Red Hat Developer Toolset version of **binutils** provides these features:

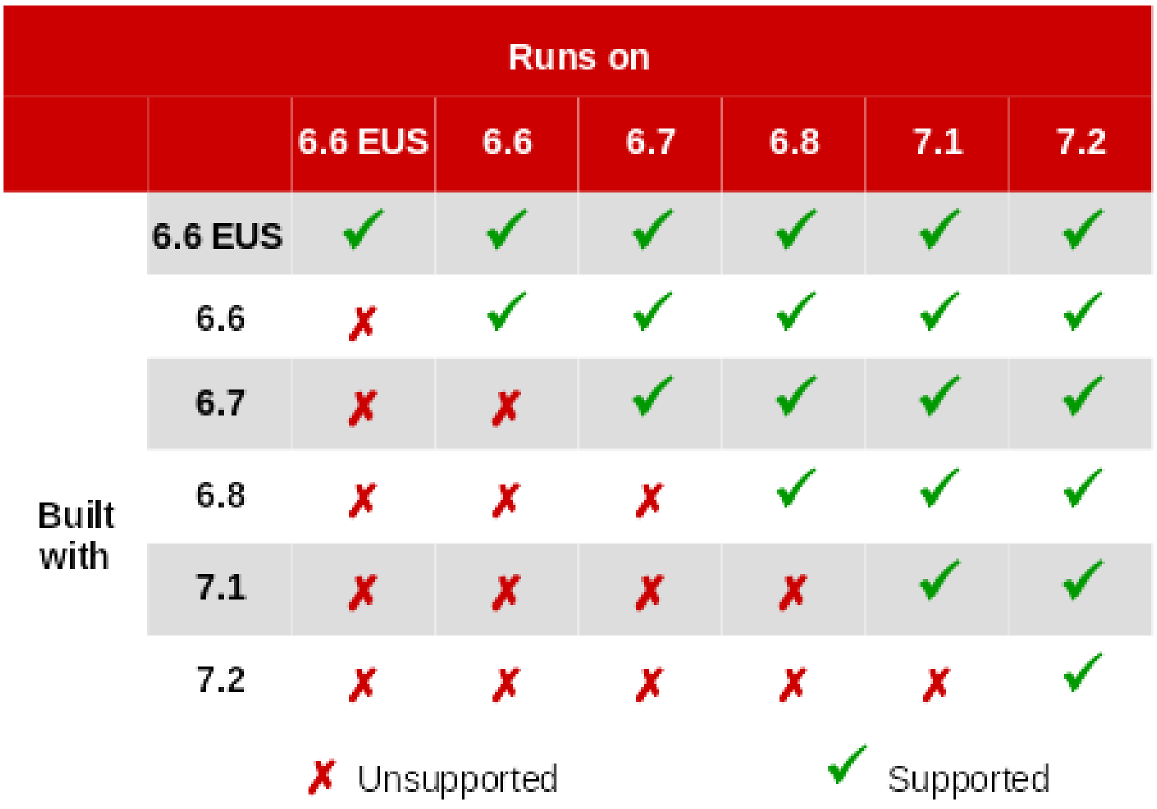
- Improved security achieved through more intensive checking of the integrity of the binary files examined by the **binutils** tools. Therefore, it is much harder to make these tools crash or to attempt to read memory that does not belong to them.

For a full list of changes and features introduced in this release, see Appendix B, *Changes in Version 4.1*.

1.3. COMPATIBILITY

Red Hat Developer Toolset 4.1 is available for Red Hat Enterprise Linux 6 and 7 for 64-bit Intel and AMD architectures. Figure 1.1, “Red Hat Developer Toolset 4.1 Compatibility Matrix” illustrates the support for binaries built with Red Hat Developer Toolset on a certain version of Red Hat Enterprise Linux when those binaries are run on various other versions of this system.

For ABI compatibility information, see Section 3.2.4, “C++ Compatibility”.



		Runs on					
		6.6 EUS	6.6	6.7	6.8	7.1	7.2
Built with	6.6 EUS	✓	✓	✓	✓	✓	✓
	6.6	✗	✓	✓	✓	✓	✓
	6.7	✗	✗	✓	✓	✓	✓
	6.8	✗	✗	✗	✓	✓	✓
	7.1	✗	✗	✗	✗	✓	✓
	7.2	✗	✗	✗	✗	✗	✓

✗ Unsupported ✓ Supported

Figure 1.1. Red Hat Developer Toolset 4.1 Compatibility Matrix

1.4. GETTING ACCESS TO RED HAT DEVELOPER TOOLSET

Red Hat Developer Toolset is an offering that is distributed as a part of the Red Hat Software Collections content set, which is available to customers with Red Hat Enterprise Linux 6 and 7 subscriptions listed at https://access.redhat.com/solutions/472793. Depending on the subscription management service with which you registered your Red Hat Enterprise Linux system, you can either enable Red Hat Developer Toolset by using the Red Hat Subscription Management, or by using RHN Classic.

For detailed instructions on how to enable Red Hat Software Collections (and thus gain access to Red Hat Developer Toolset) using RHN Classic or Red Hat Subscription Management, see the respective section below. For information on how to register your system with one of these subscription management services, see the Red Hat Subscription Management collection of guides.

1.4.1. Using Red Hat Subscription Management

If your system is registered with Red Hat Subscription Management, complete the following steps to attach a subscription that provides access to the repository for Red Hat Software Collections (which includes Red Hat Developer Toolset), and then enable that repository:

1. Determine the pool ID of a subscription that provides Red Hat Software Collections (and thus also Red Hat Developer Toolset). To do so, type the following at a shell prompt as `root` to display a list of all subscriptions that are available for your system:

   ```
   subscription-manager list --available
   ```

   For each available subscription, this command displays its name, unique identifier, expiration date, and other details related to your subscription. The pool ID is listed on a line beginning with `Pool ID`.

For a complete list of subscriptions that provide access to Red Hat Developer Toolset, see https://access.redhat.com/solutions/472793.

2. Attach the appropriate subscription to your system by running the following command as **`root`**:

   ```
   subscription-manager attach --pool=pool_id
   ```

   Replace *pool_id* with the pool ID you determined in the previous step. To verify the list of subscriptions your system has currently attached, at any time, run as **`root`**:

   ```
   subscription-manager list --consumed
   ```

3. Determine the exact name of the Red Hat Software Collections repository. To do so, type the following at a shell prompt as **`root`** to retrieve repository metadata and to display a list of available **Yum** repositories:

   ```
   subscription-manager repos --list
   ```

   The repository names depend on the specific version of Red Hat Enterprise Linux you are using and are in the following format:

   ```
   rhel-variant-rhscl-version-rpms
   rhel-variant-rhscl-version-debug-rpms
   rhel-variant-rhscl-version-source-rpms
   ```

   In addition, certain packages, such as devtoolset-4-gcc-plugin-devel, depend on packages that are only available in the **Optional** channel. The repository names with these packages use the following format:

   ```
   rhel-version-variant-optional-rpms
   rhel-version-variant-optional-debug-rpms
   rhel-version-variant-optional-source-rpms
   ```

   For both the regular repositories and optional repositories, replace *variant* with the Red Hat Enterprise Linux system variant (**`server`** or **`workstation`**), and *version* with the Red Hat Enterprise Linux system version (**`6-eus`**, **`6`**, or **`7`**).

4. Enable the repositories from step no. 3 by running the following command as **`root`**:

   ```
   subscription-manager repos --enable repository
   ```

   Replace *repository* with the name of the repository to enable.

Once the subscription is attached to the system, you can install Red Hat Developer Toolset as described in Section 1.5, "Installing Red Hat Developer Toolset". For more information on how to register your system using Red Hat Subscription Management and associate it with subscriptions, see the Red Hat Subscription Management collection of guides.

1.4.2. Using RHN Classic

If you are running Red Hat Enterprise Linux 6, and your system is registered with RHN Classic, complete the following steps to subscribe to Red Hat Software Collections (which includes Red Hat Developer Toolset):

1. Determine the exact name of the Red Hat Software Collections channel. To do so, type the following at a shell prompt as **root** to display a list of all channels that are available to you:

   ```
   rhn-channel --available-channels
   ```

   The name of the channel depends on the specific version of Red Hat Enterprise Linux you are using and is in the following format:

   ```
   rhel-x86_64-variant-version-rhscl-1
   ```

   In addition, certain packages, such as devtoolset-4-gcc-plugin-devel, depend on packages that are only available in the **Optional** channel. The name of this channel uses the following format:

   ```
   rhel-x86_64-variant-optional-6
   ```

   Replace *variant* with the Red Hat Enterprise Linux system variant (**server** or **workstation**).

2. Subscribe the system to the channels from step no. 1 by running the following command as **root**:

   ```
   rhn-channel --add --channel=channel_name
   ```

   Replace *channel_name* with the name of the channel to enable.

3. To verify the list of channels you are subscribed to, at any time, run as **root**:

   ```
   rhn-channel --list
   ```

Once the system is subscribed, you can install Red Hat Developer Toolset as described in Section 1.5, “Installing Red Hat Developer Toolset”. For more information on how to register your system with RHN Classic, see the Red Hat Subscription Management collection of guides.

1.5. INSTALLING RED HAT DEVELOPER TOOLSET

Red Hat Developer Toolset is distributed as a collection of RPM packages that can be installed, updated, uninstalled, and inspected by using the standard package management tools that are included in Red Hat Enterprise Linux. Note that a valid subscription that provides access to the Red Hat Software Collections content set is required in order to install Red Hat Developer Toolset on your system. For detailed instructions on how to associate your system with an appropriate subscription and get access to Red Hat Developer Toolset, see Section 1.4, “Getting Access to Red Hat Developer Toolset”.

IMPORTANT

Before installing Red Hat Developer Toolset, install all available Red Hat Enterprise Linux updates.

1.5.1. Installing All Available Components

To install all components that are included in Red Hat Developer Toolset, install the devtoolset-4 package by typing the following at a shell prompt as **root**:

```
yum install devtoolset-4
```

This installs the **Eclipse** development environment, all development, debugging, and performance monitoring tools, and other dependent packages to the system. Alternatively, you can choose to install only a selected package group as described in Section 1.5.2, “Installing Individual Package Groups”.

NOTE

Note that since Red Hat Developer Toolset 3.0, the scl-utils package is not a part of Red Hat Developer Toolset, which is a change from preceding versions where the `scl` utility was installed along with the Red Hat Developer Toolset software collection.

1.5.2. Installing Individual Package Groups

To make it easier to install only certain components, such as the integrated development environment or the software development toolchain, Red Hat Developer Toolset is distributed with a number of meta packages that allow you to install selected package groups as described in Table 1.2, “Red Hat Developer Toolset Meta Packages”.

Table 1.2. Red Hat Developer Toolset Meta Packages

Package Name	Description	Installed Components
devtoolset-4-ide	Integrated Development Environment	Eclipse
devtoolset-4-perftools	Performance monitoring tools	SystemTap, Valgrind, OProfile, Dyninst
devtoolset-4-toolchain	Development and debugging tools	GCC, GDB, binutils, elfutils, dwz, memstomp, strace, ltrace

To install any of these meta packages, type the following at a shell prompt as **root**:

```
yum install package_name
```

Replace *package_name* with a space-separated list of meta packages you want to install. For example, to install only the **Eclipse** development environment and packages that depend on it, type as **root**:

```
~]# yum install devtoolset-4-ide
```

Alternatively, you can choose to install all available components as described in Section 1.5.1, “Installing All Available Components”.

1.5.3. Installing Optional Packages

Red Hat Developer Toolset is distributed with a number of optional packages that are not installed by default. To list all Red Hat Developer Toolset packages that are available to you but not installed on your system, type the following command at a shell prompt:

```
yum list available devtoolset-4-\*
```

To install any of these optional packages, run as **`root`**:

```
yum install package_name
```

Replace *package_name* with a space-separated list of packages that you want to install. For example, to install the devtoolset-4-gdb-gdbserver and devtoolset-4-gdb-doc packages, type:

```
~]# yum install devtoolset-4-gdb-gdbserver devtoolset-4-gdb-doc
```

1.5.4. Installing Debugging Information

To install debugging information for any of the Red Hat Developer Toolset packages, make sure that the yum-utils package is installed and run the following command as **`root`**:

```
debuginfo-install package_name
```

For example, to install debugging information for the devtoolset-4-dwz package, type:

```
~]# debuginfo-install devtoolset-4-dwz
```

Note that in order to use this command, you need to have access to the repository with these packages. If your system is registered with Red Hat Subscription Management, enable the **`rhel-`*`variant`*`-rhscl-`*`version`*`-debug-rpms`** repository as described in Section 1.4.1, “Using Red Hat Subscription Management”. If your system is registered with RHN Classic, subscribe the system to the **`rhel-x86_64-`*`variant`*`-`*`version`*`-debuginfo`** channel as described in Section 1.4.2, “Using RHN Classic”. For more information on how to get access to debuginfo packages, see https://access.redhat.com/site/solutions/9907.

NOTE

The devtoolset-4-*package_name*-debuginfo packages can conflict with the corresponding packages from the base Red Hat Enterprise Linux system or from other versions of Red Hat Developer Toolset. This conflict also occurs in a multilib environment, where 64-bit debuginfo packages conflict with 32-bit debuginfo packages. Manually uninstall the conflicting debuginfo packages prior to installing Red Hat Developer Toolset 4.1 and install only relevant debuginfo packages when necessary.

1.6. UPDATING RED HAT DEVELOPER TOOLSET

1.6.1. Updating to a Minor Version

When a new minor version of Red Hat Developer Toolset is available, run the following command as **`root`** to update your Red Hat Enterprise Linux installation:

```
yum update
```

This updates all packages on your Red Hat Enterprise Linux system, including the Red Hat Developer Toolset versions of the **Eclipse** development environment, development, debugging, and performance monitoring tools, and other dependent packages.

IMPORTANT

Use of Red Hat Developer Toolset requires the removal of any earlier pre-release versions of it. Additionally, it is not possible to update to Red Hat Developer Toolset 4.1 from a pre-release version of Red Hat Developer Toolset, including beta releases. If you have previously installed any pre-release version of Red Hat Developer Toolset, uninstall it from your system as described in Section 1.7, “Uninstalling Red Hat Developer Toolset” and install the new version as documented in Section 1.5, “Installing Red Hat Developer Toolset”.

1.6.2. Updating to a Major Version

When a new major version of Red Hat Developer Toolset is available, you can install it in parallel with the previous version. For detailed instructions on how to install Red Hat Developer Toolset on your system, see Section 1.5, “Installing Red Hat Developer Toolset” .

1.7. UNINSTALLING RED HAT DEVELOPER TOOLSET

To uninstall Red Hat Developer Toolset packages from your system, type the following at a shell prompt as `root`:

```
yum remove devtoolset-4\* libasan libatomic libcilkrts libitm liblsan
libtsan libubsan
```

This removes the **GNU Compiler Collection**, **GNU Debugger**, **binutils**, and other packages that are a part of Red Hat Developer Toolset from the system.

NOTE

Red Hat Developer Toolset 4.1 for Red Hat Enterprise Linux 7 no longer includes the `libatomic` and `libitm` libraries, which the above command attempts to remove, because they are not required for a proper function of Red Hat Developer Toolset components on that system. Nevertheless, the above command works as expected even on Red Hat Enterprise Linux 7.

Note that the uninstallation of the tools provided by Red Hat Developer Toolset does not affect the Red Hat Enterprise Linux system versions of these tools.

1.8. USING RED HAT DEVELOPER TOOLSET CONTAINER IMAGES

Docker-formatted container images can be used to run Red Hat Developer Toolset components inside virtual software containers, thus isolating them from the host system and allowing for their rapid deployment. This section describes how to install and use pre-built container images with Red Hat Developer Toolset components, as well as how to obtain Red Hat Developer Toolset *Dockerfiles* for building custom container images and how to use the resulting images.

NOTE

The docker package, which contains the **Docker** daemon, command line tool, and other necessary components for building and using docker-formatted container images, is currently only available for the Server variant of the Red Hat Enterprise Linux 7 product. Docker-formatted container images cannot be run on Red Hat Enterprise Linux 6, and Red Hat Developer Toolset Dockerfiles are not distributed for Red Hat Enterprise Linux 6.

Follow the instructions outlined at Getting Docker in RHEL 7 to set up an environment for building and using docker-formatted container images.

1.8.1. Using Pre-Built Container Images

Pre-built docker-formatted container images are available that contain selected toolchain and perftools components of Red Hat Developer Toolset. This section describes how to obtain the pre-built Red Hat Developer Toolset docker-formatted container images and how to run Red Hat Developer Toolset components using these images.

The following images are available from the Red Hat Container Registry at `registry.access.redhat.com`:

- rhscl/devtoolset-4-toolchain-rhel7

  The image contains the following Red Hat Developer Toolset components:

  - devtoolset-4-gcc
  - devtoolset-4-gcc-c++
  - devtoolset-4-gcc-fortran
  - devtoolset-4-gdb

- rhscl/devtoolset-4-perftools-rhel7

  The image contains all Red Hat Developer Toolset components included in the devtoolset-4-perftools metapackage:

  - devtoolset-4-oprofile
  - devtoolset-4-systemtap
  - devtoolset-4-valgrind
  - devtoolset-4-dyninst

1.8.1.1. Pulling Pre-Built Container Images from the Registry

To pull a pre-built Red Hat Developer Toolset docker-formatted container image from the registry to your local machine, run the following command as `root`:

```
docker pull image_name
```

Substitute the *image_name* parameter with the name of the container image. For example, to pull the image containing selected Red Hat Developer Toolset toolchain components (rhscl/devtoolset-4-toolchain-rhel7), run the following command as `root`:

```
~]# docker pull rhscl/devtoolset-4-toolchain-rhel7
```

1.8.1.2. Running Red Hat Developer Toolset Tools from Pre-Built Container Images

To display general usage information for pre-built Red Hat Developer Toolset docker-formatted container images that you have already pulled to your local machine (see Section 1.8.1.1, “Pulling Pre-Built Container Images from the Registry”), run the following command as `root`:

```
docker run image_name usage
```

To launch an interactive shell within a pre-built docker-formatted container image, run the following command as `root`:

```
docker run -ti image_name /bin/bash -l
```

In both of the above commands, substitute the *image_name* parameter with the name of the container image you pulled to your local system and now want to use.

For example, to launch an interactive shell within the container image with selected toolchain components, run the following command as `root`:

```
~]# docker run -ti rhscl/devtoolset-4-toolchain-rhel7 /bin/bash -l
```

Example 1.1. Using GCC in the Pre-Built Red Hat Developer Toolset Toolchain Image

This example illustrates how to obtain and launch the pre-built docker-formatted container image with selected toolchain components of the Red Hat Developer Toolset and how to run the **gcc** compiler within that image.

1. Make sure you have a **Docker** environment set up properly on your system by following instructions at Getting Docker in RHEL 7.

2. Pull the pre-built toolchain Red Hat Developer Toolset container image from the official Red Hat Container Registry:

   ```
   ~]# docker pull rhscl/devtoolset-4-toolchain-rhel7
   ```

3. To launch the container image with an interactive shell, issue the following command:

   ```
   ~]# docker run -ti rhscl/devtoolset-4-toolchain-rhel7 /bin/bash -l
   ```

4. To launch the container as a regular (non-root) user, use the **sudo** command. To map a directory from the host system to the container file system, include the **-v** (or **--volume**) option in the **docker** command:

   ```
   ~]$ sudo docker run -v ~/Source:/src -ti rhscl/devtoolset-4-
   toolchain-rhel7 /bin/bash -l
   ```

In the above command, the host's **~/Source/** directory is mounted as the **/src/** directory within the container.

5. Once you are in the container's interactive shell, you can run Red Hat Developer Toolset tools as expected. For example, to verify the version of the **gcc** compiler, run:

```
bash-4.2$ gcc -v
[...]
gcc version 5.2.1 20150716 (Red Hat 5.2.1-1) (GCC)
```

1.8.2. Using Container Images Built from Dockerfiles

Dockerfiles are available for selected Red Hat Developer Toolset components. Dockerfiles are text files that contain instructions for automated building of docker-formatted container images. This section describes how to obtain Red Hat Developer Toolset Dockerfiles, how to use them to build docker-formatted container images, and how to run Red Hat Developer Toolset components using the resulting container images.

Red Hat Developer Toolset 4.1 for Red Hat Enterprise Linux 7 is shipped with the following Dockerfiles:

- devtoolset-4-dyninst
- devtoolset-4-elfutils
- devtoolset-4-oprofile
- devtoolset-4-systemtap
- devtoolset-4-toolchain
- devtoolset-4-valgrind
- devtoolset-4

1.8.2.1. Obtaining Dockerfiles

The Red Hat Developer Toolset Dockerfiles are provided by the devtoolset-4-dockerfiles package. The package contains individual Dockerfiles for building docker-formatted container images with individual components and a meta-Dockerfile for building a docker-formatted container image with all the components offered. To be able to use the Dockerfiles, install this package by executing:

```
~]# yum install devtoolset-4-dockerfiles
```

1.8.2.2. Building Container Images

Change to the directory where the Dockerfile is installed and run the following command as **root**:

```
docker build -t image_name
```

Replace *image_name* with the desired name for the new image.

Example 1.2. Building a Container Image with a Red Hat Developer Toolset Component

To build a docker-formatted container image for deploying the **elfutils** tools in a container, follow the instructions below:

1. Make sure you have a **Docker** environment set up properly on your system by following instructions at Getting Docker in RHEL 7.

2. Install the package containing the Red Hat Developer Toolset Dockerfiles:

   ```
   ~]# yum install devtoolset-4-dockerfiles
   ```

3. Determine where the Dockerfile for the required component is located:

   ```
   ~]# rpm -ql devtoolset-4-dockerfiles | grep "elfutils/Dockerfile"
   /opt/rh/devtoolset-4/root/usr/share/devtoolset-4-
   dockerfiles/rhel7/devtoolset-4-elfutils/Dockerfile
   ```

4. Change to the directory where the required Dockerfile is installed:

   ```
   ~]# cd /opt/rh/devtoolset-4/root/usr/share/devtoolset-4-
   dockerfiles/rhel7/devtoolset-4-elfutils/
   ```

5. Build the container image:

   ```
   ~]# docker build -t devtoolset-4-elfutils-7 .
   ```

   Replace *devtoolset-4-elfutils-7* with the name you wish to assign to your resulting container image.

1.8.2.3. Running Red Hat Developer Toolset Tools from Custom-Built Container Images

To display general usage information for images built from Red Hat Developer Toolset Dockerfiles (see Section 1.8.2.2, “Building Container Images”), run the following command as `root`:

```
docker run image_name container-usage
```

To launch an interactive shell within a docker-formatted container image you built, run the following command as `root`:

```
docker run -ti image_name /bin/bash -l
```

In both of the above commands, substitute the *image_name* parameter with the name of the container image you chose when building it.

Example 1.3. Using elfutils in a Custom-Built Red Hat Developer Toolset Image

This example illustrates how to launch a custom-built docker-formatted container image (built in Example 1.2, “Building a Container Image with a Red Hat Developer Toolset Component”) with the **elfutils** component and how to run the `eu-size` tool within that image.

1. To launch the container image with an interactive shell, issue the following command:

```
~]# docker run -ti devtoolset-4-elfutils-7 /bin/bash -l
```

2. To launch the container as a regular (non-root) user, use the **sudo** command. To map a directory from the host system to the container file system, include the **-v** (or **--volume**) option in the **docker** command:

   ```
   ~]$ sudo docker run -v ~/Source:/src -ti devtoolset-4-elfutils-7
   /bin/bash -l
   ```

   In the above command, the host's **~/Source/** directory is mounted as the **/src/** directory within the container.

3. Once you are in the container's interactive shell, you can run Red Hat Developer Toolset tools as expected. For example, to verify the version of the **eu-size** tool, run:

   ```
   bash-4.2$ eu-size -V
   size (elfutils) 0.163
   [...]
   ```

Using the SystemTap Tool from Container Images
When using the **SystemTap** tool from a container image (built using the Dockerfile supplied by the devtoolset-4-dockerfiles package or from the pre-built **perftools** image), additional configuration is required, and the container needs to be run with special command-line options.

The following three conditions need to be met:

1. The image needs to be run with super-user privileges. To do this, run the image using the following command:

   ```
   ~]$ docker run --ti --privileged --ipc=host --net=host --pid=host
   devtoolset-4-systemtap /bin/bash -l
   ```

   The above command assumes that you named the image devtoolset-4-systemtap when you built it from the Dockerfile (**/opt/rh/devtoolset-4/root/usr/share/devtoolset-4-dockerfiles/rhel7/devtoolset-4-systemtap/Dockerfile**).

   To use the **perftools** image, substitute the image name for devtoolset-4-perftools-rhel7 in the above command.

2. The following kernel packages need to be installed in the container:

kernel
kernel-devel
kernel-debuginfo

   The version and release numbers of the above packages must match the version and release numbers of the kernel running on the host system. Run the following command to determine the version and release numbers of the hosts system's kernel:

```
~]$ uname -r
3.10.0-229.14.1.el7.x86_64
```

Note that the kernel-debuginfo package is only available from the *Debug* channel. Enable the **rhel-7-server-debug-rpms** repository as described in Section 1.4.1, "Using Red Hat Subscription Management". For more information on how to get access to debuginfo packages, see https://access.redhat.com/site/solutions/9907.

To install the required packages with the correct version, use the **yum** package manager and the output of the **uname** command. For example, to install the correct version of the kernel package, run the following command as **root**:

```
~]# yum install -y kernel-$(uname -r)
```

3. Save the container to a reusable image by executing the **docker commit** command. For example, to save the custom-built **SystemTap** container:

```
~]$ docker commit devtoolset-4-systemtap-$(uname -r)
```

1.9. ADDITIONAL RESOURCES

For more information about Red Hat Developer Toolset and Red Hat Enterprise Linux, see the resources listed below.

Online Documentation

- Red Hat Subscription Management collection of guides — The Red Hat Subscription Management collection of guides provides detailed information on how to manage subscriptions on Red Hat Enterprise Linux.

- Red Hat Developer Toolset 4.1 Release Notes — The *Release Notes* for Red Hat Developer Toolset 4.1 contain more information.

- Red Hat Enterprise Linux 6 Developer Guide and Red Hat Enterprise Linux 7 Developer Guide — The *Developer Guides* for Red Hat Enterprise Linux 6 and 7 provide more information on the **Eclipse** IDE, libraries and runtime support, compiling and building, debugging, and profiling on these systems.

- Red Hat Enterprise Linux 6 Installation Guide and Red Hat Enterprise Linux 7 Installation Guide — The *Installation Guides* for Red Hat Enterprise Linux 6 an 7 explain how to obtain, install, and update the system.

- Red Hat Enterprise Linux 6 Deployment Guide — The *Deployment Guide* for Red Hat Enterprise Linux 6 documents relevant information regarding the deployment, configuration, and administration of Red Hat Enterprise Linux 6.

- Red Hat Enterprise Linux 7 System Administrator's Guide — The *System Administrator's Guide* for Red Hat Enterprise Linux 7 documents relevant information regarding the deployment, configuration, and administration of Red Hat Enterprise Linux 7.

- Get Started with Docker Formatted Container Images on Red Hat Systems — The guide contains a comprehensive overview of information about building and using docker-formatted container images on Red Hat Enterprise Linux 7 and Red Hat Enterprise Linux Atomic.

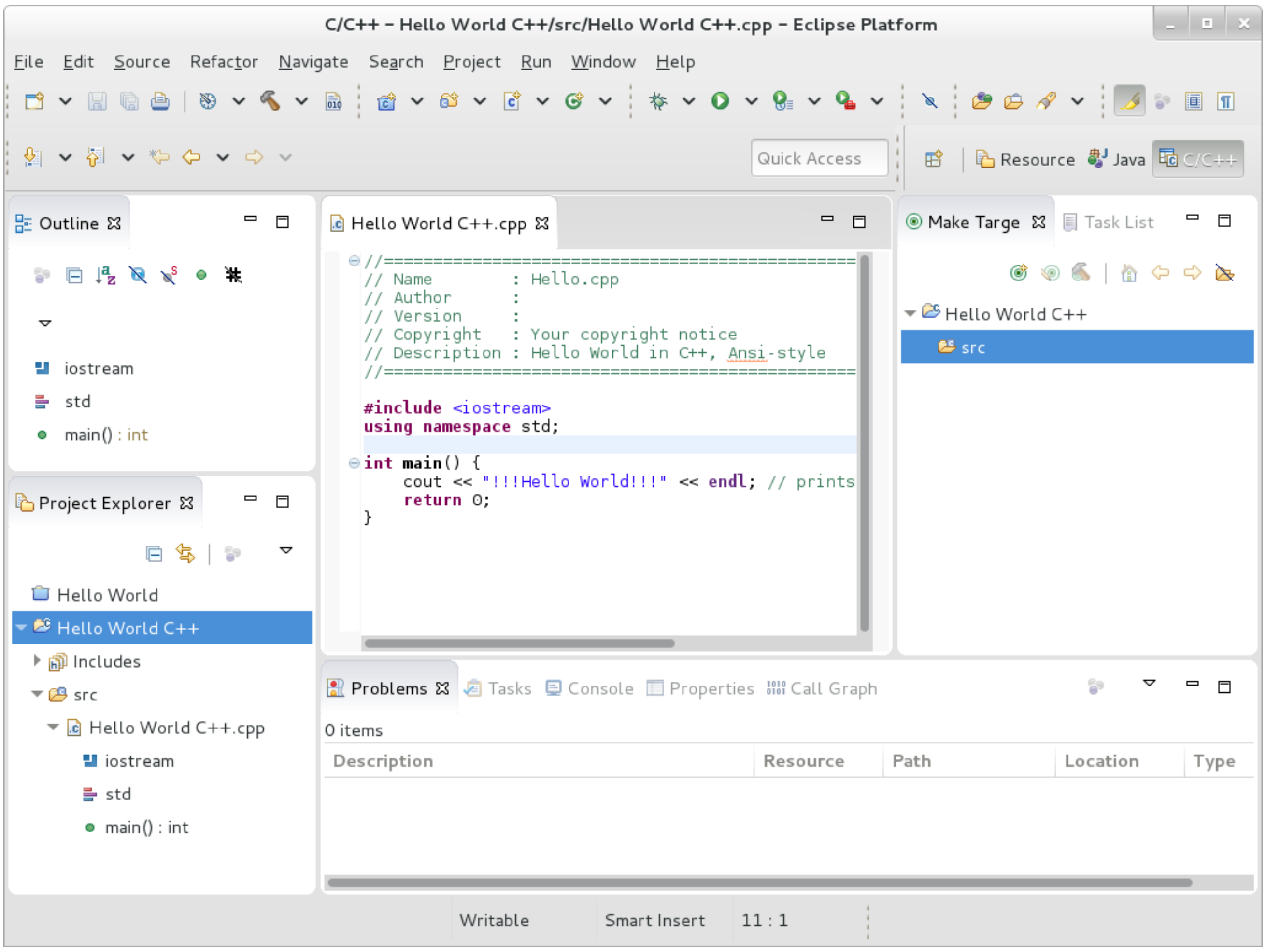
See Also

- Appendix B, *Changes in Version 4.1* provides a list of changes and improvements over the version of the GNU Compiler Collection and GNU Debugger in the previous version of Red Hat Developer Toolset.

PART II. INTEGRATED DEVELOPMENT ENVIRONMENTS

CHAPTER 2. ECLIPSE

Eclipse is a powerful development environment that provides tools for each phase of the development process. It integrates a variety of disparate tools into a unified environment to create a rich development experience, provides a fully configurable user interface, and features a pluggable architecture that allows for an extension in a variety of ways. For instance, the **Valgrind** plug-in allows programmers to perform memory profiling, otherwise performed on the command line, through the **Eclipse** user interface.



```
//==================================================
// Name        : Hello.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//==================================================

#include <iostream>
using namespace std;

int main() {
	cout << "!!!Hello World!!!" << endl; // prints
	return 0;
}
```

```
0 items
Description                Resource    Path    Location    Type
```

```
Writable    Smart Insert    11 : 1
```

Figure 2.1. Sample Eclipse Session

Eclipse provides a graphical development environment alternative to traditional interaction with command line tools and as such, it is a welcome alternative to developers who do not want to use the command line interface. The traditional, mostly command line-based Linux tools suite (such as **gcc** or **gdb**) and **Eclipse** offer two distinct approaches to programming.

Red Hat Developer Toolset is distributed with **Eclipse 4.5.2**, which is based on the Eclipse Foundation's Mars release train. Note that if you intend to develop applications for Red Hat JBoss Middleware or require support for OpenShift Tools, it is recommended that you use Red Hat JBoss Developer Studio.

Table 2.1. Eclipse Components Included in Red Hat Developer Toolset

Package	Description
devtoolset-4-eclipse-cdt	The C/C++ Development Tooling (CDT), which provides features and plug-ins for development in C and C++.

Package	Description
devtoolset-4-eclipse-emf	The Eclipse Modeling Framework (EMF), which allows you to build applications based on a structured data model.
devtoolset-4-eclipse-gef	The Graphical Editing Framework (GEF), which allows you to create a rich graphical editor from an existing application model.
devtoolset-4-eclipse-rse	The Remote System Explorer (RSE) framework, which allows you to work with remote systems from Eclipse.
devtoolset-4-eclipse-jgit	JGit, a Java implementation of the **Git** revision control system.
devtoolset-4-eclipse-egit	EGit, a team provider for **Eclipse** that provides features and plug-ins for interaction with Git repositories.
devtoolset-4-eclipse-mylyn	Mylyn, a task management system for **Eclipse**.
devtoolset-4-eclipse-pydev	A full featured Python IDE for **Eclipse**.
devtoolset-4-eclipse-ptp	A subset of the PTP project providing support for synchronized projects.
devtoolset-4-eclipse-pde	The Plugin Development Environment for developing **Eclipse** plugins.
devtoolset-4-eclipse-tests	**Eclipse** tests.
devtoolset-4-eclipse-linuxtools	A meta package for Linux-specific **Eclipse** plug-ins.
devtoolset-4-eclipse-remote	The Remote Services plug-in, which provides an extensible remote-services framework.
devtoolset-4-eclipse-changelog[a]	The ChangeLog plug-in, which allows you to create and maintain changelog files.
devtoolset-4-eclipse-gcov[a]	The GCov plug-in, which integrates the **GCov** test coverage program with **Eclipse**.
devtoolset-4-eclipse-gprof[a]	The Gprof plug-in, which integrates the **Gprof** performance analysis utility with **Eclipse**.
devtoolset-4-eclipse-manpage[a]	The Man Page plug-in, which allows you to view manual pages in **Eclipse**.
devtoolset-4-eclipse-oprofile[a]	The OProfile plug-in, which integrates **OProfile** with **Eclipse**.
devtoolset-4-eclipse-perf[a]	The Perf plug-in, which integrates the **perf** tool with **Eclipse**.

Package	Description
devtoolset-4-eclipse-rpm-editor[a]	The Eclipse Spec File Editor, which allows you to maintain RPM spec files.
devtoolset-4-eclipse-systemtap[a]	The SystemTap plug-in, which integrates **SystemTap** with **Eclipse**.
devtoolset-4-eclipse-valgrind[a]	The Valgrind plug-in, which integrates **Valgrind** with **Eclipse**.

[a] This package is installed as a dependency of devtoolset-4-eclipse-linuxtools.

2.1. INSTALLING ECLIPSE

In Red Hat Developer Toolset, the **Eclipse** development environment is provided as a collection of RPM packages and is automatically installed with the devtoolset-4-ide package as described in Section 1.5, "Installing Red Hat Developer Toolset". For a list of available components, see Table 2.1, "Eclipse Components Included in Red Hat Developer Toolset".

NOTE

The Red Hat Developer Toolset version of Eclipse fully supports C, C++, and Java development, but does *not* provide support for the Fortran programming language.

2.2. USING ECLIPSE

To start the Red Hat Developer Toolset version of **Eclipse**, either select **Applications** → **Programming** → **DTS Eclipse** from the panel, or type the following at a shell prompt:

```
scl enable devtoolset-4 'eclipse'
```

During its startup, **Eclipse** prompts you to select a *workspace*, that is, a directory in which you want to store your projects. You can either use `~/workspace/`, which is the default option, or click the **Browse** button to browse your file system and select a custom directory. Additionally, you can select the `Use this as the default and do not ask again` check box to prevent **Eclipse** from displaying this dialog box the next time you run this development environment. When you are done, click the **OK** button to confirm the selection and proceed with the startup.

2.2.1. Using the Red Hat Developer Toolset Toolchain

To use the Red Hat Developer Toolset version of **Eclipse** with support for the **GNU Compiler Collection** and **binutils** from Red Hat Developer Toolset, make sure that the devtoolset-4-toolchain package is installed and run the application as described in Section 2.2, "Using Eclipse". Red Hat Developer Toolset **Eclipse** uses the Red Hat Developer Toolset toolchain by default.

For detailed instructions on how to install the devtoolset-4-toolchain package in your system, see Section 1.5, "Installing Red Hat Developer Toolset".

IMPORTANT

If you are working on a project that you previously built with the Red Hat Enterprise Linux version of the **GNU Compiler Collection**, make sure that you discard all previous build results. To do so, open the project in **Eclipse** and select **Project** → **Clean** from the menu.

2.2.2. Using the Red Hat Enterprise Linux Toolchain

To use the Red Hat Developer Toolset version of **Eclipse** with support for the toolchain distributed with Red Hat Enterprise Linux, change the configuration of the project to use absolute paths to the Red Hat Enterprise Linux system versions of **gcc**, **g++**, and **as**.

To configure **Eclipse** to explicitly use the Red Hat Enterprise Linux system versions of the tools for the current project, complete the following steps:

1. In the C/C++ perspective, choose **Project** → **Properties** from the main menu bar to open the project properties.

2. In the menu on the left-hand side of the dialog box, click **C/C++ Build** → **Settings**.

3. Select the **`Tool Settings`** tab.

4. If you are working on a C project:

    1. select **GCC C Compiler** or **Cross GCC Compiler** and change the value of the **Command** field to:

        ```
        /usr/bin/gcc
        ```

    2. select **GCC C Linker** or **Cross GCC Linker** and change the value of the **Command** field to:

        ```
        /usr/bin/gcc
        ```

    3. select **GCC Assembler** or **Cross GCC Assembler** and change the value of the **Command** field to:

        ```
        /usr/bin/as
        ```

    If you are working on a C++ project:

    1. select **GCC C++ Compiler** or **Cross G++ Compiler** and change the value of the **Command** field to:

        ```
        /usr/bin/g++
        ```

    2. select **GCC C Compiler** or **Cross GCC Compiler** and change the value of the **Command** field to:

        ```
        /usr/bin/gcc
        ```

    3. select **GCC C++ Linker** or **Cross G++ Linker** and change the value of the **Command** field to:

```
/usr/bin/g++
```

4. select **GCC Assembler** or **Cross GCC Assembler** and change the value of the **Command** field to:

```
/usr/bin/as
```

5. Click the **OK** button to save the configuration changes.

2.3. ADDITIONAL RESOURCES

A detailed description of **Eclipse** and all its features is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

- **Eclipse** includes a built-in **Help** system, which provides extensive documentation for each integrated feature and tool. This greatly decreases the initial time investment required for new developers to become fluent in its use. The use of this Help section is detailed in the *Red Hat Enterprise Linux Developer Guide* linked below.

See Also

- Section B.1, "Changes in Eclipse" provides a comprehensive list of features and improvements over the **Eclipse** development environment included in the previous release of Red Hat Developer Toolset.

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.

- Chapter 3, *GNU Compiler Collection (GCC)* provides information on how to compile programs written in C, C++, and Fortran on the command line.

PART III. DEVELOPMENT TOOLS

CHAPTER 3. GNU COMPILER COLLECTION (GCC)

The **GNU Compiler Collection**, commonly abbreviated **GCC**, is a portable compiler suite with support for a wide selection of programming languages.

Red Hat Developer Toolset is distributed with **GCC 5.3.1**. This version is more recent than the version included in Red Hat Enterprise Linux and provides a number of bug fixes and enhancements.

3.1. GNU C COMPILER

3.1.1. Installing the C Compiler

In Red Hat Developer Toolset, the GNU C compiler is provided by the devtoolset-4-gcc package and is automatically installed with devtoolset-4-toolchain as described in Section 1.5, “Installing Red Hat Developer Toolset”.

3.1.2. Using the C Compiler

To compile a C program on the command line, run the **gcc** compiler as follows:

```
scl enable devtoolset-4 'gcc -o output_file source_file...'
```

This creates a binary file named *output_file* in the current working directory. If the **-o** option is omitted, the compiler creates a file named **a.out** by default.

When you are working on a project that consists of several source files, it is common to compile an object file for each of the source files first and then link these object files together. This way, when you change a single source file, you can recompile only this file without having to compile the entire project. To compile an object file on the command line, run the following command:

```
scl enable devtoolset-4 'gcc -o object_file -c source_file'
```

This creates an object file named *object_file*. If the **-o** option is omitted, the compiler creates a file named after the source file with the **.o** file extension. To link object files together and create a binary file, run:

```
scl enable devtoolset-4 'gcc -o output_file object_file...'
```

Note that you can execute any command using the **scl** utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset **gcc** as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of **gcc** you are using at any point, type the following at a shell prompt:

```
which gcc
```

Red Hat Developer Toolset's **gcc** executable path will begin with `/opt`. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset **gcc**:

```
gcc -v
```

IMPORTANT

Some newer library features are statically linked into applications built with Red Hat Developer Toolset to support execution on multiple versions of Red Hat Enterprise Linux. This adds a small additional security risk as normal Red Hat Enterprise Linux errata would not change this code. If the need for developers to rebuild their applications due to such an issue arises, Red Hat will signal this via a security erratum. Developers are strongly advised not to statically link their entire application for the same reasons.

NOTE

The Red Hat Developer Toolset 4.1 version of **GCC** supports Cilk+, an extension to the C and C++ languages for parallel programming, which can be enabled using the **`-fcilkplus`** option. A runtime library, **`libcilkrts`**, is included in this release to support Cilk+. The **`libcilkrts`** library has been a part of Red Hat Enterprise Linux since version 7.2, but the package is not included in all supported Red Hat Enterprise Linux releases. To enable dynamic linkage of binaries and libraries built with Red Hat Developer Toolset 4.1 **GCC** using Cilk+ features on supported Red Hat Enterprise Linux releases that do not contain **`libcilkrts`**, install the **`libcilkrts.so`** shared library from Red Hat Developer Toolset 4.1 with such binaries or libraries.

Example 3.1. Compiling a C Program on the Command Line

Consider a source file named **`hello.c`** with the following contents:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello, World!\n");
  return 0;
}
```

To compile this source code on the command line by using the **gcc** compiler from Red Hat Developer Toolset, type:

```
~]$ scl enable devtoolset-4 'gcc -o hello hello.c'
```

This creates a new binary file called **`hello`** in the current working directory.

3.1.3. Running a C Program

When **gcc** compiles a program, it creates an executable binary file. To run this program on the command line, change to the directory with the executable file and type:

```
./file_name
```

Example 3.2. Running a C Program on the Command Line

Assuming that you have successfully compiled the `hello` binary file as shown in Example 3.1, "Compiling a C Program on the Command Line", you can run it by typing the following at a shell prompt:

```
~]$ ./hello
Hello, World!
```

3.2. GNU C++ COMPILER

3.2.1. Installing the C++ Compiler

In Red Hat Developer Toolset, the GNU C++ compiler is provided by the devtoolset-4-gcc-c++ package and is automatically installed with the devtoolset-4-toolchain package as described in Section 1.5, "Installing Red Hat Developer Toolset".

3.2.2. Using the C++ Compiler

To compile a C++ program on the command line, run the **g++** compiler as follows:

```
scl enable devtoolset-4 'g++ -o output_file source_file...'
```

This creates a binary file named *output_file* in the current working directory. If the `-o` option is omitted, the **g++** compiler creates a file named `a.out` by default.

When you are working on a project that consists of several source files, it is common to compile an object file for each of the source files first and then link these object files together. This way, when you change a single source file, you can recompile only this file without having to compile the entire project. To compile an object file on the command line, run the following command:

```
scl enable devtoolset-4 'g++ -o object_file -c source_file'
```

This creates an object file named *object_file*. If the `-o` option is omitted, the **g++** compiler creates a file named after the source file with the `.o` file extension. To link object files together and create a binary file, run:

```
scl enable devtoolset-4 'g++ -o output_file object_file...'
```

Note that you can execute any command using the **scl** utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset **g++** as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of **g++** you are using at any point, type the following at a shell prompt:

```
which g++
```

Red Hat Developer Toolset's **g++** executable path will begin with */opt*. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset **g++**:

```
g++ -v
```

IMPORTANT

Some newer library features are statically linked into applications built with Red Hat Developer Toolset to support execution on multiple versions of Red Hat Enterprise Linux. This adds a small additional security risk as normal Red Hat Enterprise Linux errata would not change this code. If the need for developers to rebuild their applications due to such an issue arises, Red Hat will signal this via a security erratum. Developers are strongly advised not to statically link their entire application for the same reasons.

Example 3.3. Compiling a C++ Program on the Command Line

Consider a source file named **hello.cpp** with the following contents:

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[]) {
  cout << "Hello, World!" << endl;
  return 0;
}
```

To compile this source code on the command line by using the **g++** compiler from Red Hat Developer Toolset, type:

```
~]$ scl enable devtoolset-4 'g++ -o hello hello.cpp'
```

This creates a new binary file called **hello** in the current working directory.

3.2.3. Running a C++ Program

When **g++** compiles a program, it creates an executable binary file. To run this program on the command line, change to the directory with the executable file and type:

```
./file_name
```

Example 3.4. Running a C++ Program on the Command Line

Assuming that you have successfully compiled the **hello** binary file as shown in Example 3.3, "Compiling a C++ Program on the Command Line", you can run it by typing the following at a shell prompt:

```
~]$ ./hello
Hello, World!
```

3.2.4. C++ Compatibility

All compilers from Red Hat Enterprise Linux versions 5, 6, and 7 and from Red Hat Developer Toolset versions 1, 2, 3, and 4 in any **-std** mode are compatible with any other of those compilers in C++98 mode. A compiler in C++11 or C++14 mode is only guaranteed to be compatible with another compiler in C++11 or C++14 mode if they are from the same release series (for example from Red Hat Developer Toolset 4.x).

3.2.4.1. C++ ABI

Because the upstream GCC community development does not guarantee C++11 ABI compatibility across major versions of GCC, the same applies to use of C++11 with Red Hat Developer Toolset. Consequently, using the **-std=c++11** option is supported in Red Hat Developer Toolset 3.x only when all C++ objects compiled with that flag have been built using the same major version of Red Hat Developer Toolset. The mixing of objects, binaries and libraries, built by the Red Hat Enterprise Linux 6 or 7 system toolchain GCC using the **-std=c++0x** or **-std=gnu++0x** flags, with those built with the **-std=c++11** or **-std=gnu++11** flags using the GCC in Red Hat Developer Toolset is explicitly not supported.

As later major versions of Red Hat Developer Toolset may use a later major release of GCC, forward-compatibility of objects, binaries, and libraries built with the **-std=c++11** or **-std=gnu++11** options cannot be guaranteed, and so is not supported.

The default language standard setting for Red Hat Developer Toolset is C++98. Any C++98-compliant binaries or libraries built in this default mode (or explicitly with **-std=c++98**) can be freely mixed with binaries and shared libraries built by the Red Hat Enterprise Linux 6 or 7 system toolchain GCC. Red Hat recommends use of this default **-std=c++98** mode for production software development.

IMPORTANT

Use of C++11 features in your application requires careful consideration of the above ABI compatibility information.

Aside from the C++11 ABI, discussed above, the Red Hat Enterprise Linux Application Compatibility Specification is unchanged for Red Hat Developer Toolset. When mixing objects built with Red Hat Developer Toolset with those built with the Red Hat Enterprise Linux 6 or 7 toolchain (particularly

`.o/.a` files), the Red Hat Developer Toolset toolchain should be used for any linkage. This ensures any newer library features provided only by Red Hat Developer Toolset are resolved at link-time.

A new standard mangling for SIMD vector types has been added to avoid name clashes on systems with vectors of varying length. By default the compiler still uses the old mangling, but emits aliases with the new mangling on targets that support strong aliases. `-Wabi` will now display a warning about code that uses the old mangling.

3.3. GNU FORTRAN COMPILER

3.3.1. Installing the Fortran Compiler

In Red Hat Developer Toolset, the GNU Fortran compiler is provided by the devtoolset-4-gcc-gfortran package and is automatically installed with devtoolset-4-toolchain as described in Section 1.5, "Installing Red Hat Developer Toolset".

3.3.2. Using the Fortran Compiler

To compile a Fortran program on the command line, run the `gfortran` compiler as follows:

```
scl enable devtoolset-4 'gfortran -o output_file source_file...'
```

This creates a binary file named *output_file* in the current working directory. If the `-o` option is omitted, the compiler creates a file named `a.out` by default.

When you are working on a project that consists of several source files, it is common to compile an object file for each of the source files first and then link these object files together. This way, when you change a single source file, you can recompile only this file without having to compile the entire project. To compile an object file on the command line, run the following command:

```
scl enable devtoolset-4 'gfortran -o object_file -c source_file'
```

This creates an object file named *object_file*. If the `-o` option is omitted, the compiler creates a file named after the source file with the `.o` file extension. To link object files together and create a binary file, run:

```
scl enable devtoolset-4 'gfortran -o output_file object_file...'
```

Note that you can execute any command using the `scl` utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset `gfortran` as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of **`gfortran`** you are using at any point, type the following at a shell prompt:

```
which gfortran
```

Red Hat Developer Toolset's **`gfortran`** executable path will begin with **`/opt`**. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset **`gfortran`**:

```
gfortran -v
```

IMPORTANT

Some newer library features are statically linked into applications built with Red Hat Developer Toolset to support execution on multiple versions of Red Hat Enterprise Linux. This adds a small additional security risk as normal Red Hat Enterprise Linux errata would not change this code. If the need for developers to rebuild their applications due to such an issue arises, Red Hat will signal this via a security erratum. Developers are strongly advised not to statically link their entire application for the same reasons.

Example 3.5. Compiling a Fortran Program on the Command Line

Consider a source file named **`hello.f`** with the following contents:

```
program hello
  print *, "Hello, World!"
end program hello
```

To compile this source code on the command line by using the **`gfortran`** compiler from Red Hat Developer Toolset, type:

```
~]$ scl enable devtoolset-4 'gfortran -o hello hello.f'
```

This creates a new binary file called **`hello`** in the current working directory.

3.3.3. Running a Fortran Program

When **`gfortran`** compiles a program, it creates an executable binary file. To run this program on the command line, change to the directory with the executable file and type:

```
./file_name
```

Example 3.6. Running a Fortran Program on the Command Line

Assuming that you have successfully compiled the **`hello`** binary file as shown in Example 3.5, "Compiling a Fortran Program on the Command Line", you can run it by typing the following at a shell prompt:

```
~]$ ./hello
 Hello, World!
```

3.4. ADDITIONAL RESOURCES

A detailed description of the GNU Compiler Collections and its features is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

- **gcc**(1) — The manual page for the **gcc** compiler provides detailed information on its usage; with few exceptions, **g++** accepts the same command line options as **gcc**. To display the manual page for the version included in Red Hat Developer Toolset, type:

  ```
  scl enable devtoolset-4 'man gcc'
  ```

- **gfortran**(1) — The manual page for the **gfortran** compiler provides detailed information on its usage. To display the manual page for the version included in Red Hat Developer Toolset, type:

  ```
  scl enable devtoolset-4 'man gfortran'
  ```

- *C++ Standard Library Documentation*— Documentation on the C++ standard library can be optionally installed by typing the following at a shell prompt as **root**:

  ```
  yum install devtoolset-4-libstdc++-docs
  ```

  Once installed, HTML documentation is available at **/opt/rh/devtoolset-4/root/usr/share/doc/devtoolset-4-libstdc++-docs-5.3.1/html/index.html**.

Online Documentation

- Red Hat Enterprise Linux 6 Developer Guide and Red Hat Enterprise Linux 7 Developer Guide — The *Developer Guides* for Red Hat Enterprise Linux 6 and 7 provide in-depth information about **GCC**.

- Using the GNU Compiler Collection — The official GCC manual provides an in-depth description of the GNU compilers and their usage.

- The GNU C++ Library — The GNU C++ library documentation provides detailed information about the GNU implementation of the standard C++ library.

- The GNU Fortran Compiler — The GNU Fortran compiler documentation provides detailed information on **gfortran**'s usage.

See Also

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.

- Chapter 2, *Eclipse* provides a general introduction to the **Eclipse** development environment, and describes how to use it with the tools from Red Hat Developer Toolset.

- Chapter 4, *binutils* explains how to use the **binutils**, a collection of binary tools to inspect and manipulate object files and binaries.
- Chapter 5, *elfutils* explains how to use **elfutils**, a collection of binary tools to inspect and manipulate ELF files.
- Chapter 6, *dwz* explains how to use **dwz** to optimize DWARF debugging information contained in ELF shared libraries and ELF executables for size.
- Chapter 7, *GNU Debugger (GDB)* provides information on how to debug programs written in C, C++, and Fortran.

CHAPTER 4. BINUTILS

binutils is a collection of various binary tools, such as the **GNU linker**, **GNU assembler**, and other utilities that allow you to inspect and manipulate object files and binaries. See Table 4.1, “Tools Included in binutils for Red Hat Developer Toolset” for a complete list of binary tools that are distributed with the Red Hat Developer Toolset version of **binutils**.

Red Hat Developer Toolset is distributed with **binutils 2.25.1**. This version is more recent than the version included in Red Hat Enterprise Linux and the previous release of Red Hat Developer Toolset and provides bug fixes and enhancements.

Table 4.1. Tools Included in binutils for Red Hat Developer Toolset

Name	Description
`addr2line`	Translates addresses into file names and line numbers.
`ar`	Creates, modifies, and extracts files from archives.
`as`	The GNU assembler.
`c++filt`	Decodes mangled C++ symbols.
`dwp`	Combines DWARF object files into a single DWARF package file.
`elfedit`	Examines and edits ELF files.
`gprof`	Display profiling information.
`ld`	The GNU linker.
`ld.bfd`	An alternative to the GNU linker.
`ld.gold`	Another alternative to the GNU linker.
`nm`	Lists symbols from object files.
`objcopy`	Copies and translates object files.
`objdump`	Displays information from object files.
`ranlib`	Generates an index to the contents of an archive to make access to this archive faster.
`readelf`	Displays information about ELF files.
`size`	Lists section sizes of object or archive files.
`strings`	Displays printable character sequences in files.

Name	Description
`strip`	Discards all symbols from object files.

4.1. INSTALLING BINUTILS

In Red Hat Developer Toolset, **binutils** are provided by the devtoolset-4-binutils package and are automatically installed with devtoolset-4-toolchain as described in Section 1.5, “Installing Red Hat Developer Toolset”.

4.2. USING THE GNU ASSEMBLER

To produce an object file from an assembly language program, run the **as** tool as follows:

```
scl enable devtoolset-4 'as [option...] -o object_file source_file'
```

This creates an object file named *object_file* in the current working directory.

Note that you can execute any command using the `scl` utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset **as** as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of **as** you are using at any point, type the following at a shell prompt:

```
which as
```

Red Hat Developer Toolset's **as** executable path will begin with `/opt`. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset **as**:

```
as -v
```

4.3. USING THE GNU LINKER

To create an executable binary file or a library from object files, run the `ld` tool as follows:

```
scl enable devtoolset-4 'ld [option...] -o output_file object_file...'
```

This creates a binary file named *output_file* in the current working directory. If the `-o` option is omitted, the compiler creates a file named `a.out` by default.

Note that you can execute any command using the `scl` utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset `ld` as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of `ld` you are using at any point, type the following at a shell prompt:

```
which ld
```

Red Hat Developer Toolset's `ld` executable path will begin with `/opt`. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset `ld`:

```
ld -v
```

4.4. USING OTHER BINARY TOOLS

The **binutils** provide many binary tools other than a linker and an assembler. For a complete list of these tools, see Table 4.1, "Tools Included in binutils for Red Hat Developer Toolset" .

To execute any of the tools that are a part of binutils, run the command as follows:

```
scl enable devtoolset-4 'tool [option...] file_name'
```

See Table 4.1, "Tools Included in binutils for Red Hat Developer Toolset" for a list of tools that are distributed with **binutils**. For example, to use the `objdump` tool to inspect an object file, type:

```
scl enable devtoolset-4 'objdump [option...] object_file'
```

Note that you can execute any command using the `scl` utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset binary tools as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of **binutils** you are using at any point, type the following at a shell prompt:

```
which objdump
```

Red Hat Developer Toolset's `objdump` executable path will begin with `/opt`. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset `objdump`:

```
objdump -v
```

4.5. ADDITIONAL RESOURCES

A detailed description of **binutils** is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

- as(1), ld(1), addr2line(1), ar(1), c++filt(1), dwp(1), elfedit(1), gprof(1), nm(1), objcopy(1), objdump(1), ranlib(1), readelf(1), size(1), strings(1), strip(1), — Manual pages for various **binutils** tools provide more information about their respective usage. To display a manual page for the version included in Red Hat Developer Toolset, type:

  ```
  scl enable devtoolset-4 'man tool'
  ```

Online Documentation

- Documentation for binutils — The **binutils** documentation provides an in-depth description of the binary tools and their usage.

See Also

- Section B.2, "Changes in binutils" provides a comprehensive list of features and improvements over the Red Hat Enterprise Linux 6.8 version of **binutils** and the version distributed in the 4.0 release of Red Hat Developer Toolset.

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.

- Chapter 5, *elfutils* explains how to use **elfutils**, a collection of binary tools to inspect and manipulate ELF files.

- Chapter 3, *GNU Compiler Collection (GCC)* provides information on how to compile programs written in C, C++, and Fortran.

CHAPTER 5. ELFUTILS

elfutils is a collection of various binary tools, such as **`eu-objdump`**, **`eu-readelf`**, and other utilities that allow you to inspect and manipulate ELF files. See Table 5.1, “Tools Included in elfutils for Red Hat Developer Toolset” for a complete list of binary tools that are distributed with the Red Hat Developer Toolset version of **elfutils**.

Red Hat Developer Toolset is distributed with **elfutils 0.166**. This version is more recent than the version included the previous release of Red Hat Developer Toolset and provides some bug fixes and enhancements.

Table 5.1. Tools Included in elfutils for Red Hat Developer Toolset

Name	Description
`eu-addr2line`	Translates addresses into file names and line numbers.
`eu-ar`	Creates, modifies, and extracts files from archives.
`eu-elfcmp`	Compares relevant parts of two ELF files for equality.
`eu-elflint`	Verifies that ELF files are compliant with the *generic ABI* (gABI) and *processor-specific supplement ABI* (psABI) specification.
`eu-findtextrel`	Locates the source of text relocations in files.
`eu-make-debug-archive`	Creates an offline archive for debugging.
`eu-nm`	Lists symbols from object files.
`eu-objdump`	Displays information from object files.
`eu-ranlib`	Generates an index to the contents of an archive to make access to this archive faster.
`eu-readelf`	Displays information about ELF files.
`eu-size`	Lists section sizes of object or archive files.
`eu-stack`	A new utility for unwinding processes and cores.
`eu-strings`	Displays printable character sequences in files.
`eu-strip`	Discards all symbols from object files.
`eu-unstrip`	Combines stripped files with separate symbols and debug information.

5.1. INSTALLING ELFUTILS

In Red Hat Developer Toolset, **elfutils** is provided by the devtoolset-4-elfutils package and is automatically installed with devtoolset-4-toolchain as described in Section 1.5, “Installing Red Hat Developer Toolset”.

5.2. USING ELFUTILS

To execute any of the tools that are part of **elfutils**, run the command as follows:

```
scl enable devtoolset-4 'tool [option...] file_name'
```

See Table 5.1, “Tools Included in elfutils for Red Hat Developer Toolset” for a list of tools that are distributed with **elfutils**. For example, to use the **eu-objdump** tool to inspect an object file, type:

```
scl enable devtoolset-4 'eu-objdump [option...] object_file'
```

Note that you can execute any command using the **scl** utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset binary tools as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of **elfutils** you are using at any point, type the following at a shell prompt:

```
which eu-objdump
```

Red Hat Developer Toolset's **eu-objdump** executable path will begin with **/opt**. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset **eu-objdump**:

```
eu-objdump -V
```

5.3. ADDITIONAL RESOURCES

A detailed description of **elfutils** is beyond the scope of this book. For more information, see the resources listed below.

See Also

- Section B.3, “Changes in elfutils” provides a comprehensive list of features and improvements over the version distributed in the previous release of Red Hat Developer Toolset.
- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.
- Chapter 3, *GNU Compiler Collection (GCC)* provides information on how to compile programs written in C, C++, and Fortran.
- Chapter 4, *binutils* explains how to use the **binutils**, a collection of binary tools to inspect and manipulate object files and binaries.

- Chapter 6, *dwz* explains how to use **dwz** to optimize DWARF debugging information contained in ELF shared libraries and ELF executables for size.

CHAPTER 6. DWZ

dwz is a command line tool that attempts to optimize DWARF debugging information contained in ELF shared libraries and ELF executables for size. To do so, **dwz** replaces DWARF information representation with equivalent smaller representation where possible and reduces the amount of duplication by using techniques from *Appendix E* of the *DWARF Standard*.

Red Hat Developer Toolset is distributed with **dwz 0.12**.

6.1. INSTALLING DWZ

In Red Hat Developer Toolset, the **dwz** utility is provided by the devtoolset-4-dwz package and is automatically installed with devtoolset-4-toolchain as described in Section 1.5, “Installing Red Hat Developer Toolset”.

6.2. USING DWZ

To optimize DWARF debugging information in a binary file, run the **dwz** tool as follows:

```
scl enable devtoolset-4 'dwz [option...] file_name'
```

Note that you can execute any command using the **scl** utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset **dwz** as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of **dwz** you are using at any point, type the following at a shell prompt:

```
which dwz
```

Red Hat Developer Toolset's **dwz** executable path will begin with **/opt**. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset **dwz**:

```
dwz -v
```

6.3. ADDITIONAL RESOURCES

A detailed description of **dwz** and its features is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

- dwz(1) — The manual page for the **dwz** utility provides detailed information on its usage. To display the manual page for the version included in Red Hat Developer Toolset, type:

  ```
  scl enable devtoolset-4 'man dwz'
  ```

-

See Also

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.
- Chapter 3, *GNU Compiler Collection (GCC)* provides information on how to compile programs written in C, C++, and Fortran.
- Chapter 4, *binutils* explains how to use the **binutils**, a collection of binary tools to inspect and manipulate object files and binaries.
- Chapter 5, *elfutils* explains how to use **elfutils**, a collection of binary tools to inspect and manipulate ELF files.

PART IV. DEBUGGING TOOLS

CHAPTER 7. GNU DEBUGGER (GDB)

The **GNU Debugger**, commonly abbreviated as **GDB**, is a command line tool that can be used to debug programs written in various programming languages. It allows you to inspect memory within the code being debugged, control the execution state of the code, detect the execution of particular sections of code, and much more.

Red Hat Developer Toolset is distributed with **GDB 7.10**. This version is more recent than the version included in Red Hat Enterprise Linux and the previous release of Red Hat Developer Toolset and provides some enhancements and numerous bug fixes.

7.1. INSTALLING THE GNU DEBUGGER

In Red Hat Developer Toolset, the **GNU Debugger** is provided by the devtoolset-4-gdb package and is automatically installed with devtoolset-4-toolchain as described in Section 1.5, “Installing Red Hat Developer Toolset”.

7.2. PREPARING A PROGRAM FOR DEBUGGING

Compiling Programs with Debugging Information

To compile a C program with debugging information that can be read by the **GNU Debugger**, make sure the **gcc** compiler is run with the **-g** option. To do so on the command line, use a command in the following form:

```
scl enable devtoolset-4 'gcc -g -o output_file input_file...'
```

Similarly, to compile a C++ program with debugging information, run:

```
scl enable devtoolset-4 'g++ -g -o output_file input_file...'
```

Example 7.1. Compiling a C Program With Debugging Information

Consider a source file named **fibonacci.c** that has the following contents:

```
#include <stdio.h>
#include <limits.h>

int main (int argc, char *argv[]) {
  unsigned long int a = 0;
  unsigned long int b = 1;
  unsigned long int sum;

  while (b < LONG_MAX) {
    printf("%ld ", b);
    sum = a + b;
    a = b;
    b = sum;
  }

  return 0;
}
```

To compile this program on the command line using **GCC** from Red Hat Developer Toolset with debugging information for the **GNU Debugger**, type:

```
~]$ scl enable devtoolset-4 'gcc -g -o fibonacci fibonacci.c'
```

This creates a new binary file called `fibonacci` in the current working directory.

Installing Debugging Information for Existing Packages

To install debugging information for a package that is already installed on the system, type the following at a shell prompt as `root`:

```
debuginfo-install package_name
```

Note that the yum-utils package must be installed for the `debuginfo-install` utility to be available on your system.

Example 7.2. Installing Debugging Information for the glibc Package

To install debugging information for the glibc package, type:

```
~]# debuginfo-install glibc
Loaded plugins: product-id, refresh-packagekit, subscription-manager
--> Running transaction check
---> Package glibc-debuginfo.x86_64 0:2.17-105.el7 will be installed
...
```

7.3. RUNNING THE GNU DEBUGGER

To run the **GNU Debugger** on a program you want to debug, type the following at a shell prompt:

```
scl enable devtoolset-4 'gdb file_name'
```

This starts the **gdb** debugger in interactive mode and displays the default prompt, `(gdb)`. To quit the debugging session and return to the shell prompt, run the following command at any time:

```
quit
```

Note that you can execute any command using the `scl` utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset **gdb** as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of **gdb** you are using at any point, type the following at a shell prompt:

```
which gdb
```

Red Hat Developer Toolset's **gdb** executable path will begin with `/opt`. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset **gdb**:

```
gdb -v
```

Example 7.3. Running the gdb Utility on the fibonacci Binary File

Assuming that you have successfully compiled the `fibonacci` binary file as shown in Example 7.1, "Compiling a C Program With Debugging Information", you can start debugging it with **gdb** by typing the following at a shell prompt:

```
~]$ scl enable devtoolset-4 'gdb fibonacci'
GNU gdb (GDB) Red Hat Enterprise Linux (7.10-20.el7)
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show
copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb)
```

7.4. LISTING SOURCE CODE

To view the source code of the program you are debugging, run the following command:

```
list
```

Before you start the execution of the program you are debugging, **gdb** displays the first ten lines of the source code, and any subsequent use of this command lists another ten lines. Once you start the execution, **gdb** displays the lines that are surrounding the line on which the execution stops, typically when you set a breakpoint.

You can also display the code that is surrounding a particular line. To do so, run the command in the following form:

```
list [file_name:]line_number
```

Similarly, to display the code that is surrounding the beginning of a particular function, run:

```
list [file_name:]function_name
```

Note that you can change the number of lines the **list** command displays by running the following command:

```
set listsize number
```

Example 7.4. Listing the Source Code of the fibonacci Binary File

The **fibonacci.c** file listed in Example 7.1, “Compiling a C Program With Debugging Information” has exactly 17 lines. Assuming that you have compiled it with debugging information and you want the **gdb** utility to be capable of listing the entire source code, you can run the following command to change the number of listed lines to 20:

```
(gdb) set listsize 20
```

You can now display the entire source code of the file you are debugging by running the **list** command with no additional arguments:

```
(gdb) list
1       #include <stdio.h>
2       #include <limits.h>
3
4       int main (int argc, char *argv[]) {
5         unsigned long int a = 0;
6         unsigned long int b = 1;
7         unsigned long int sum;
8
9         while (b < LONG_MAX) {
10          printf("%ld ", b);
11          sum = a + b;
12          a = b;
13          b = sum;
14        }
15
16        return 0;
17      }
```

7.5. SETTING BREAKPOINTS

Setting a New Breakpoint

To set a new breakpoint at a certain line, run the following command:

```
break [file_name:]line_number
```

You can also set a breakpoint on a certain function:

```
break [file_name:]function_name
```

Example 7.5. Setting a New Breakpoint

Assuming that you have compiled the **fibonacci.c** file listed in Example 7.1, “Compiling a C Program With Debugging Information” with debugging information, you can set a new breakpoint at line 10 by running the following command:

```
(gdb) break 10
Breakpoint 1 at 0x4004e5: file fibonacci.c, line 10.
```

Listing Breakpoints

To display a list of currently set breakpoints, run the following command:

```
info breakpoints
```

Example 7.6. Listing Breakpoints

Assuming that you have followed the instructions in Example 7.5, “Setting a New Breakpoint” , you can display the list of currently set breakpoints by running the following command:

```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000004004e5 in main at
fibonacci.c:10
```

Deleting Existing Breakpoints

To delete a breakpoint that is set at a certain line, run the following command:

```
clear line_number
```

Similarly, to delete a breakpoint that is set on a certain function, run:

```
clear function_name
```

Example 7.7. Deleting an Existing Breakpoint

Assuming that you have compiled the **fibonacci.c** file listed in Example 7.1, “Compiling a C Program With Debugging Information” with debugging information, you can set a new breakpoint at line 7 by running the following command:

```
(gdb) break 7
Breakpoint 2 at 0x4004e3: file fibonacci.c, line 7.
```

To remove this breakpoint, type:

```
(gdb) clear 7
Deleted breakpoint 2
```

7.6. STARTING EXECUTION

To start an execution of the program you are debugging, run the following command:

```
run
```

If the program accepts any command line arguments, you can provide them as arguments to the **run** command:

```
run argument…
```

The execution stops when the first breakpoint (if any) is reached, when an error occurs, or when the program terminates.

Example 7.8. Executing the fibonacci Binary File

Assuming that you have followed the instructions in Example 7.5, "Setting a New Breakpoint" , you can execute the **fibonacci** binary file by running the following command:

```
(gdb) run
Starting program: /home/john/fibonacci

Breakpoint 1, main (argc=1, argv=0x7fffffffe4d8) at fibonacci.c:10
10          printf("%ld ", b);
```

7.7. DISPLAYING CURRENT VALUES

The **gdb** utility allows you to display the value of almost anything that is relevant to the program, from a variable of any complexity to a valid expression or even a library function. However, the most common task is to display the value of a variable.

To display the current value of a certain variable, run the following command:

```
print variable_name
```

Example 7.9. Displaying the Current Values of Variables

Assuming that you have followed the instructions in Example 7.8, "Executing the fibonacci Binary File" and the execution of the **fibonacci** binary stopped after reaching the breakpoint at line 10, you can display the current values of variables **a** and **b** as follows:

```
(gdb) print a
$1 = 0
(gdb) print b
$2 = 1
```

7.8. CONTINUING EXECUTION

To resume the execution of the program you are debugging after it reached a breakpoint, run the following command:

```
continue
```

The execution stops again when another breakpoint is reached. To skip a certain number of breakpoints (typically when you are debugging a loop), you can run the **`continue`** command in the following form:

```
continue number
```

The **gdb** utility also allows you to stop the execution after executing a single line of code. To do so, run:

```
step
```

Finally, you can execute a certain number of lines by using the **`step`** command in the following form:

```
step number
```

Example 7.10. Continuing the Execution of the fibonacci Binary File

Assuming that you have followed the instructions in Example 7.8, “Executing the fibonacci Binary File”, and the execution of the **`fibonacci`** binary stopped after reaching the breakpoint at line 10, you can resume the execution by running the following command:

```
(gdb) continue
Continuing.

Breakpoint 1, main (argc=1, argv=0x7fffffffe4d8) at fibonacci.c:10
10          printf("%ld ", b);
```

The execution stops the next time the breakpoint is reached. To execute the next three lines of code, type:

```
(gdb) step 3
13          b = sum;
```

This allows you to verify the current value of the **sum** variable before it is assigned to **b**:

```
(gdb) print sum
$3 = 2
```

7.9. ADDITIONAL RESOURCES

A detailed description of the **GNU Debugger** and all its features is beyond the scope of this book. For more information, see the resources listed below.

Online Documentation

- Red Hat Enterprise Linux 6 Developer Guide and Red Hat Enterprise Linux 7 Developer Guide – The *Developer Guides* for Red Hat Enterprise Linux 6 and 7 provide more information on the

GNU Debugger and debugging.

- GDB Documentation — The official **GDB** documentation includes the *GDB User Manual* and other reference material.

See Also

- Section A.6, “Changes in GDB” provides a comprehensive list of features and improvements over the Red Hat Enterprise Linux system version of the **GNU Debugger** and the version distributed in the previous release of Red Hat Developer Toolset.

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.

- Chapter 2, *Eclipse* provides a general introduction to the **Eclipse** development environment, and describes how to use it with the tools from Red Hat Developer Toolset.

- Chapter 3, *GNU Compiler Collection (GCC)* provides further information on how to compile programs written in C, C++, and Fortran.

- Chapter 8, *strace* documents how to use the **strace** utility to monitor system calls that a program uses and signals it receives.

- Chapter 10, *memstomp* documents how to use the **memstomp** utility to identify calls to library functions with overlapping memory regions that are not allowed by various standards.

CHAPTER 8. STRACE

strace is a diagnostic and debugging tool for the command line that can be used to trace system calls that are made and received by a running process. It records the name of each system call, its arguments, and its return value, as well as signals received by the process and other interactions with the kernel, and prints this record to standard error output or a selected file.

Red Hat Developer Toolset is distributed with **strace 4.10**.

8.1. INSTALLING STRACE

In Red Hat Enterprise Linux, the `strace` utility is provided by the devtoolset-4-strace package and is automatically installed with devtoolset-4-toolchain as described in Section 1.5, "Installing Red Hat Developer Toolset".

8.2. USING STRACE

To run the `strace` utility on a program you want to analyze, type the following at a shell prompt:

```
scl enable devtoolset-4 'strace program [argument...]'
```

Replace *program* with the name of the program you want to analyze, and *argument* with any command line options and arguments you want to supply to this program. Alternatively, you can run the utility on an already running process by using the `-p` command line option followed by the process ID:

```
scl enable devtoolset-4 'strace -p process_id'
```

Note that you can execute any command using the `scl` utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset `strace` as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of `strace` you are using at any point, type the following at a shell prompt:

```
which strace
```

Red Hat Developer Toolset's `strace` executable path will begin with `/opt`. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset `strace`:

```
strace -V
```

8.2.1. Redirecting Output to a File

By default, `strace` prints the name of each system call, its arguments and the return value to standard error output. To redirect this output to a file, use the `-o` command line option followed by the file name:

```
scl enable devtoolset-4 'strace -o file_name program [argument...]'
```

Replace *file_name* with the name of the file.

Example 8.1. Redirecting Output to a File

Consider a slightly modified version of the `fibonacci` file from Example 7.1, “Compiling a C Program With Debugging Information”. This executable file displays the Fibonacci sequence and optionally allows you to specify how many members of this sequence to list. To run the `strace` utility on this file and redirect the trace output to `fibonacci.log`, type:

```
~]$ scl enable devtoolset-4 'strace -o fibonacci.log ./fibonacci 20'
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

This creates a new plain-text file called `fibonacci.log` in the current working directory.

8.2.2. Tracing Selected System Calls

To trace only a selected set of system calls, run the `strace` utility with the `-e` command line option:

```
scl enable devtoolset-4 'strace -e expression program [argument...]'
```

Replace *expression* with a comma-separated list of system calls to trace or any of the keywords listed in Table 8.1, “Commonly Used Values of the -e Option” . For a detailed description of all available values, see the strace(1) manual page.

Table 8.1. Commonly Used Values of the -e Option

Value	Description
`file`	System calls that accept a file name as an argument.
`process`	System calls that are related to process management.
`network`	System calls that are related to networking.
`signal`	System calls that are related to signal management.
`ipc`	System calls that are related to inter-process communication (IPC).
`desc`	System calls that are related to file descriptors.

Example 8.2. Tracing Selected System Calls

Consider the `employee` file from Example 10.1, “Using memstomp”. To run the `strace` utility on this executable file and trace only the `mmap` and `munmap` system calls, type:

```
~]$ scl enable devtoolset-4 'strace -e mmap,munmap ./employee'
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
```

```
= 0x7f896c744000
mmap(NULL, 61239, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f896c735000
mmap(0x3146a00000, 3745960, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x3146a00000
mmap(0x3146d89000, 20480, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x189000) = 0x3146d89000
mmap(0x3146d8e000, 18600, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x3146d8e000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f896c734000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f896c733000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f896c732000
munmap(0x7f896c735000, 61239)           = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f896c743000
John,john@example.comDoe,
+++ exited with 0 +++
```

8.2.3. Displaying Time Stamps

To prefix each line of the trace with the exact time of the day in hours, minutes, and seconds, run the **strace** utility with the **-t** command line option:

```
scl enable devtoolset-4 'strace -t program [argument...]'
```

To also display milliseconds, supply the **-t** option twice:

```
scl enable devtoolset-4 'strace -tt program [argument...]'
```

To prefix each line of the trace with the time required to execute the respective system call, use the **-r** command line option:

```
scl enable devtoolset-4 'strace -r program [argument...]'
```

Example 8.3. Displaying Time Stamps

Consider an executable file named **pwd**. To run the **strace** utility on this file and include time stamps in the output, type:

```
~]$ scl enable devtoolset-4 'strace -tt pwd'
19:43:28.011815 execve("./pwd", ["./pwd"], [/* 36 vars */]) = 0
19:43:28.012128 brk(0)                  = 0xcd3000
19:43:28.012174 mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc869cb0000
19:43:28.012427 open("/etc/ld.so.cache", O_RDONLY) = 3
19:43:28.012446 fstat(3, {st_mode=S_IFREG|0644, st_size=61239, ...}) = 0
19:43:28.012464 mmap(NULL, 61239, PROT_READ, MAP_PRIVATE, 3, 0) =
0x7fc869ca1000
```

```
19:43:28.012483 close(3)                = 0
...
19:43:28.013410 +++ exited with 0 +++
```

8.2.4. Displaying a Summary

To display a summary of how much time was required to execute each system call, how many times were these system calls executed, and how many errors were encountered during their execution, run the **strace** utility with the **-c** command line option:

```
scl enable devtoolset-4 'strace -c program [argument...]'
```

Example 8.4. Displaying a Summary

Consider an executable file named **lsblk**. To run the **strace** utility on this file and display a trace summary, type:

```
~]$ scl enable devtoolset-4 'strace -c lsblk > /dev/null'
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 80.88    0.000055           1       106        16 open
 19.12    0.000013           0       140           munmap
  0.00    0.000000           0       148           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0       258           close
  0.00    0.000000           0        37         2 stat
...
------ ----------- ----------- --------- --------- ----------------
100.00    0.000068                  1790        35 total
```

8.3. ADDITIONAL RESOURCES

A detailed description of **strace** and its features is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

- strace(1) — The manual page for the **strace** utility provides detailed information about its usage. To display the manual page for the version included in Red Hat Developer Toolset, type:

```
scl enable devtoolset-4 'man strace'
```

See Also

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.

- Chapter 9, *ltrace* provides information on how to trace program library calls using the **ltrace** tool.

- Chapter 7, *GNU Debugger (GDB)* provides information on how to debug programs written in C, C++, and Fortran.

- Chapter 10, *memstomp* documents how to use the **memstomp** utility to identify calls to library functions with overlapping memory regions that are not allowed by various standards.

CHAPTER 9. LTRACE

ltrace is a diagnostic and debugging tool for the command line that can be used to display calls that are made to shared libraries. It uses the dynamic library hooking mechanism, which prevents it from tracing calls to statically linked libraries. **ltrace** also displays return values of the library calls. The output is printed to standard error output or to a selected file.

Red Hat Developer Toolset is distributed with **ltrace 0.7.91**. While the base version **ltrace** remains the same as in the previous release of Red Hat Developer Toolset, various enhancements and bug fixes have ported.

9.1. INSTALLING LTRACE

In Red Hat Enterprise Linux, the `ltrace` utility is provided by the devtoolset-4-ltrace package and is automatically installed with devtoolset-4-toolchain as described in Section 1.5, “Installing Red Hat Developer Toolset”.

9.2. USING LTRACE

To run the `ltrace` utility on a program you want to analyze, type the following at a shell prompt:

```
scl enable devtoolset-4 'ltrace program [argument...]'
```

Replace *program* with the name of the program you want to analyze, and *argument* with any command line options and arguments you want to supply to this program. Alternatively, you can run the utility on an already running process by using the `-p` command line option followed by the process ID:

```
scl enable devtoolset-4 'ltrace -p process_id'
```

Note that you can execute any command using the `scl` utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset `ltrace` as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of `ltrace` you are using at any point, type the following at a shell prompt:

```
which ltrace
```

Red Hat Developer Toolset's `ltrace` executable path will begin with `/opt`. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset `ltrace`:

```
ltrace -V
```

9.2.1. Redirecting Output to a File

By default, **ltrace** prints the name of each system call, its arguments and the return value to standard error output. To redirect this output to a file, use the **-o** command line option followed by the file name:

```
scl enable devtoolset-4 'ltrace -o file_name program [argument...]'
```

Replace *file_name* with the name of the file.

Example 9.1. Redirecting Output to a File

Consider a slightly modified version of the **fibonacci** file from Example 7.1, “Compiling a C Program With Debugging Information”. This executable file displays the Fibonacci sequence and optionally allows you to specify how many members of this sequence to list. To run the **ltrace** utility on this file and redirect the trace output to **fibonacci.log**, type:

```
~]$ scl enable devtoolset-4 'ltrace -o fibonacci.log ./fibonacci 20'
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

This creates a new plain-text file called **fibonacci.log** in the current working directory.

9.2.2. Tracing Selected Library Calls

To trace only a selected set of library calls, run the **ltrace** utility with the **-e** command line option:

```
scl enable devtoolset-4 'ltrace -e expression program [argument...]'
```

Replace *expression* with a chain of rules to specify the library calls to trace. The rules can consist of patterns that identify symbol names (such as **malloc** or **free**) and patterns that identify library SONAMEs (such as **libc.so**). For example, to trace call to the **malloc** and **free** function but to omit those that are done by the **libc** library, use:

```
scl enable devtoolset-4 'ltrace -e malloc+free-@libc.so* program'
```

Example 9.2. Tracing Selected Library Calls

Consider the **ls** command. To run the **ltrace** utility on this program and trace only the **opendir**, **readdir**, and **closedir** function calls, type:

```
~]$ scl enable devtoolset-4 'ltrace -e opendir+readdir+closedir ls'
ls->opendir(".")     = { 3 }
ls->readdir({ 3 })   = { 61533, "." }
ls->readdir({ 3 })   = { 131, ".." }
ls->readdir({ 3 })   = { 67185100, "BUILDROOT" }
ls->readdir({ 3 })   = { 202390772, "SOURCES" }
ls->readdir({ 3 })   = { 60249, "SPECS" }
ls->readdir({ 3 })   = { 67130110, "BUILD" }
ls->readdir({ 3 })   = { 136599168, "RPMS" }
ls->readdir({ 3 })   = { 202383274, "SRPMS" }
ls->readdir({ 3 })   = nil
ls->closedir({ 3 })  = 0
BUILD  BUILDROOT  RPMS  SOURCES  SPECS  SRPMS
+++ exited (status 0) +++
```

For a detailed description of available filter expressions, see the ltrace(1) manual page.

9.2.3. Displaying Time Stamps

To prefix each line of the trace with the exact time of the day in hours, minutes, and seconds, run the **ltrace** utility with the **-t** command line option:

```
scl enable devtoolset-4 'ltrace -t program [argument...]'
```

To also display milliseconds, supply the **-t** option twice:

```
scl enable devtoolset-4 'ltrace -tt program [argument...]'
```

To prefix each line of the trace with the time required to execute the respective system call, use the **-r** command line option:

```
scl enable devtoolset-4 'ltrace -r program [argument...]'
```

Example 9.3. Displaying Time Stamps

Consider the **pwd** command. To run the **ltrace** utility on this program and include time stamps in the output, type:

```
~]$ scl enable devtoolset-4 'ltrace -tt pwd'
13:27:19.631371 __libc_start_main([ "pwd" ] <unfinished ...>
13:27:19.632240 getenv("POSIXLY_CORRECT")                       = nil
13:27:19.632520 strrchr("pwd", '/')                             = nil
13:27:19.632786 setlocale(LC_ALL, "")                           =
"en_US.UTF-8"
13:27:19.633220 bindtextdomain("coreutils", "/usr/share/locale") =
"/usr/share/locale"
13:27:19.633471 textdomain("coreutils")                         =
"coreutils"
...
13:27:19.637110 +++ exited (status 0) +++
```

9.2.4. Displaying a Summary

To display a summary of how much time was required to execute each system call and how many times were these system calls executed, run the **ltrace** utility with the **-c** command line option:

```
scl enable devtoolset-4 'ltrace -c program [argument...]'
```

Example 9.4. Displaying a Summary

Consider the **lsblk** command. To run the **ltrace** utility on this program and display a trace summary, type:

```
~]$ scl enable devtoolset-4 'ltrace -c lsblk > /dev/null'
```

```
% time     seconds  usecs/call     calls      function
------ ----------- ----------- --------- --------------------
 53.60    0.261644      261644         1 __libc_start_main
  4.48    0.021848          58       374 mbrtowc
  4.41    0.021524          57       374 wcwidth
  4.39    0.021409          57       374 __ctype_get_mb_cur_max
  4.38    0.021359          57       374 iswprint
  4.06    0.019838          74       266 readdir64
  3.21    0.015652          69       224 strlen
...
------ ----------- ----------- --------- --------------------
100.00    0.488135                  3482 total
```

9.3. ADDITIONAL RESOURCES

A detailed description of **ltrace** and its features is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

- ltrace(1) — The manual page for the `ltrace` utility provides detailed information about its usage. To display the manual page for the version included in Red Hat Developer Toolset, type:

  ```
  scl enable devtoolset-4 'man ltrace'
  ```

Online Documentation

- ltrace for RHEL 6 and 7 — This article on the Red Hat Developer Blog offers additional in-depth information (including practical examples) on how to use **ltrace** for application debugging.

See Also

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.
- Chapter 8, *strace* provides information on how to trace program system calls using the **strace** tool.
- Chapter 7, *GNU Debugger (GDB)* provides information on how to debug programs written in C, C++, and Fortran.
- Chapter 10, *memstomp* documents how to use the **memstomp** utility to identify calls to library functions with overlapping memory regions that are not allowed by various standards.

CHAPTER 10. MEMSTOMP

memstomp is a command line tool that can be used to identify function calls with overlapping memory regions in situations when such an overlap is not permitted by various standards. It intercepts calls to the library functions listed in Table 10.1, "Function Calls Inspected by memstomp" and for each memory overlap, it displays a detailed backtrace to help you debug the problem.

Similarly to **Valgrind**, the **`memstomp`** utility inspects applications without the need to recompile them. However, it is much faster than this tool and therefore serves as a convenient alternative to it.

Red Hat Developer Toolset is distributed with **memstomp 0.1.5**.

Table 10.1. Function Calls Inspected by memstomp

Function	Description
`memcpy`	Copies *n* bytes from one memory area to another and returns a pointer to the second memory area.
`memccpy`	Copies a maximum of *n* bytes from one memory area to another and stops when a certain character is found. It either returns a pointer to the byte following the last written byte, or NULL if the given character is not found.
`mempcpy`	Copies *n* bytes from one memory area to another and returns a pointer to the byte following the last written byte.
`strcpy`	Copies a string from one memory area to another and returns a pointer to the second string.
`stpcpy`	Copies a string from one memory area to another and returns a pointer to the terminating null byte of the second string.
`strncpy`	Copies a maximum of *n* characters from one string to another and returns a pointer to the second string.
`stpncpy`	Copies a maximum of *n* characters from one string to another. It either returns a pointer to the terminating null byte of the second string, or if the string is not null-terminated, a pointer to the byte following the last written byte.
`strcat`	Appends one string to another while overwriting the terminating null byte of the second string and adding a new one at its end. It returns a pointer to the new string.
`strncat`	Appends a maximum of *n* characters from one string to another while overwriting the terminating null byte of the second string and adding a new one at its end. It returns a pointer to the new string.
`wmemcpy`	The wide-character equivalent of the **`memcpy`**() function that copies *n* wide characters from one array to another and returns a pointer to the second array.

Function	Description
wmempcpy	The wide-character equivalent of the **mempcpy**() function that copies *n* wide characters from one array to another and returns a pointer to the byte following the last written wide character.
wcscpy	The wide-character equivalent of the **strcpy**() function that copies a wide-character string from one array to another and returns a pointer to the second array.
wcsncpy	The wide-character equivalent of the **strncpy**() function that copies a maximum of *n* wide characters from one array to another and returns a pointer to the second string.
wcscat	The wide-character equivalent of the **strcat**() function that appends one wide-character string to another while overwriting the terminating null byte of the second string and adding a new one at its end. It returns a pointer to the new string.
wcsncat	The wide-character equivalent of the **strncat**() function that appends a maximum of *n* wide characters from one array to another while overwriting the terminating null byte of the second wide-character string and adding a new one at its end. It returns a pointer to the new string.

10.1. INSTALLING MEMSTOMP

In Red Hat Developer Toolset, the **memstomp** utility is provided by the devtoolset-4-memstomp package and is automatically installed with devtoolset-4-toolchain as described in Section 1.5, "Installing Red Hat Developer Toolset".

10.2. USING MEMSTOMP

To run the **memstomp** utility on a program you want to analyze, type the following at a shell prompt:

```
scl enable devtoolset-4 'memstomp program [argument...]'
```

To immediately terminate the analyzed program when a problem is detected, run the utility with the **--kill** (or **-k** for short) command line option:

```
scl enable devtoolset-4 'memstomp --kill program [argument...]'
```

The use of the **--kill** option is especially recommended if you are analyzing a multi-threaded program; the internal implementation of backtraces is not thread-safe and running the **memstomp** utility on a multi-threaded program without this command line option can therefore produce unreliable results.

Additionally, if you have compiled the analyzed program with the debugging information or this debugging information is available to you, you can use the **--debug-info** (or **-d**) command line option to produce a more detailed backtrace:

```
scl enable devtoolset-4 'memstomp --debug-info program [argument...]'
```

For detailed instructions on how to compile your program with the debugging information built in the binary file, see Section 7.2, "Preparing a Program for Debugging" . For information on how to install debugging information for any of the Red Hat Developer Toolset packages, see Section 1.5.4, "Installing Debugging Information".

Note that you can execute any command using the **scl** utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset **memstomp** as default:

```
scl enable devtoolset-4 'bash'
```

Example 10.1. Using memstomp

In the current working directory, create a source file named **employee.c** with the following contents:

```
#include <stdio.h>
#include <string.h>

#define BUFSIZE 80

int main(int argc, char *argv[]) {
  char employee[BUFSIZE] = "John,Doe,john@example.com";
  char name[BUFSIZE] = {0};
  char surname[BUFSIZE] = {0};
  char *email;
  size_t length;

  /* Extract the information: */
  memccpy(name, employee, ',', BUFSIZE);
  length = strlen(name);
  memccpy(surname, employee + length, ',', BUFSIZE);
  length += strlen(surname);
  email = employee + length;

  /* Compose the new entry: */
  strcat(employee, surname);
  strcpy(employee, name);
  strcat(employee, email);

  /* Print the result: */
  puts(employee);

  return 0;
}
```

Compile this program into a binary file named **employee** by using the following command:

```
~]$ scl enable devtoolset-4 'gcc -rdynamic -g -o employee employee.c'
```

To identify erroneous function calls with overlapping memory regions, type:

```
~]$ scl enable devtoolset-4 'memstomp --debug-info ./employee'
memstomp: 0.1.4 successfully initialized for process employee (pid
14887).


strcat(dest=0x7fff13afc265, src=0x7fff13afc269, bytes=21) overlap for
employee(14887)
        ??:0    strcpy()
        ??:0    strcpy()
        ??:0    _Exit()
        ??:0    strcat()
        employee.c:26   main()
        ??:0    __libc_start_main()
        ??:0    _start()
John,john@example.comDoe,
```

10.3. ADDITIONAL RESOURCES

A detailed description of **memstomp** and its features is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

- memstomp(1) — The manual page for the **memstomp** utility provides detailed information about its usage. To display the manual page for the version included in Red Hat Developer Toolset, type:

  ```
  scl enable devtoolset-4 'man memstomp'
  ```

See Also

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.

- Chapter 7, *GNU Debugger (GDB)* provides information on how to debug programs written in C, C++, and Fortran.

- Chapter 8, *strace* documents how to use the strace utility to monitor system calls that a program uses and signals it receives.

- Chapter 12, *Valgrind* explains how to use **Valgrind** to profile applications and detect memory errors and memory management problems, such as the use of uninitialized memory, improper allocation and freeing of memory, and the use of improper arguments in system calls.

PART V. PERFORMANCE MONITORING TOOLS

CHAPTER 11. SYSTEMTAP

SystemTap is a tracing and probing tool that allows users to monitor the activities of the entire system without needing to instrument, recompile, install, and reboot. It is programmable with a custom scripting language, which gives it expressiveness (to trace, filter, and analyze) and reach (to look into the running kernel and applications).

SystemTap can monitor various types of events, such as function calls within the kernel or applications, timers, tracepoints, performance counters, and so on. Some included example scripts produce output similar to **`netstat`**, **`ps`**, **`top`**, and **`iostat`**, others include pretty-printed function callgraph traces or tools for working around security bugs.

Red Hat Developer Toolset is distributed with **SystemTap 2.9**. This version is more recent than the version included in the previous release of Red Hat Developer Toolset and provides numerous bug fixes and enhancements.

Table 11.1. Tools Distributed with SystemTap for Red Hat Developer Toolset

Name	Description
`stap`	Translates probing instructions into C code, builds a kernel module, and loads it into a running Linux kernel.
`stapdyn`	The **Dyninst** backend for **SystemTap**.
`staprun`	Loads, unloads, attaches to, and detaches from kernel modules built with the **`stap`** utility.
`stapsh`	Serves as a remote shell for **SystemTap**.
`stap-prep`	Determines and—if possible—downloads the kernel information packages that are required to run **SystemTap**.
`stap-merge`	Merges per-CPU files. This script is automatically executed when the **`stap`** utility is executed with the **`-b`** command line option.
`stap-report`	Gathers important information about the system for the purpose of reporting a bug in **SystemTap**.
`stap-server`	A compile server, which listens for requests from **`stap`** clients.

11.1. INSTALLING SYSTEMTAP

In Red Hat Developer Toolset, **`SystemTap`** is provided by the devtoolset-4-systemtap package and is automatically installed with devtoolset-4-perftools as described in Section 1.5, “Installing Red Hat Developer Toolset”.

NOTE

The Red Hat Developer Toolset version of **SystemTap** is available for both Red Hat Enterprise Linux 6 and Red Hat Enterprise Linux 7, but some new features are only offered by the Red Hat Developer Toolset version of **SystemTap** for Red Hat Enterprise Linux 7.

In order to place instrumentation into the Linux kernel, **SystemTap** may also require installation of additional packages with debugging information. To determine which packages to install, run the **`stap-prep`** utility as follows:

```
scl enable devtoolset-4 'stap-prep'
```

Note that if you execute this command as the `root` user, the utility automatically offers the packages for installation. For more information on how to install these packages on your system, see the *Red Hat Enterprise Linux 6 SystemTap Beginners Guide* or the *Red Hat Enterprise Linux 7 SystemTap Beginners Guide*.

11.2. USING SYSTEMTAP

To execute any of the tools that are part of **SystemTap**, type the following at a shell prompt:

```
scl enable devtoolset-4 'tool [option...]'
```

See Table 11.1, “Tools Distributed with SystemTap for Red Hat Developer Toolset” for a list of tools that are distributed with **SystemTap**. For example, to run the **`stap`** tool to build an instrumentation module, type:

```
scl enable devtoolset-4 'stap [option...] argument...'
```

Note that you can execute any command using the **`scl`** utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset **SystemTap** as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of **SystemTap** you are using at any point, type the following at a shell prompt:

```
which stap
```

Red Hat Developer Toolset's **`stap`** executable path will begin with `/opt`. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset **SystemTap**:

```
stap -V
```

11.3. ADDITIONAL RESOURCES

A detailed description of **SystemTap** and its features is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

- stap(1) — The manual page for the `stap` command provides detailed information on its usage, as well as references to other related manual pages. To display the manual page for the version included in Red Hat Developer Toolset, type:

  ```
  scl enable devtoolset-4 'man stap'
  ```

- staprun(8) — The manual page for the `staprun` command provides detailed information on its usage. To display the manual page for the version included in Red Hat Developer Toolset, type:

  ```
  scl enable devtoolset-4 'man staprun'
  ```

- *SystemTap Tapset Reference Manual*— HTML documentation on the most common tapset definitions is located at `/opt/rh/devtoolset-4/root/usr/share/doc/devtoolset-4-systemtap-client-2.8/index.html`.

Online Documentation

- Red Hat Enterprise Linux 6 SystemTap Beginners Guide and Red Hat Enterprise Linux 7 SystemTap Beginners Guide — The *SystemTap Beginners Guides* for Red Hat Enterprise Linux 6 and 7 provide an introduction to **SystemTap** and its usage.

- Red Hat Enterprise Linux 6 SystemTap Tapset Reference and Red Hat Enterprise Linux 7 SystemTap Tapset Reference — The *SystemTap Tapset Reference* for Red Hat Enterprise Linux 6 and 7 provides further details about **SystemTap**.

- The SystemTap Documentation — The official **SystemTap** documentation provides further documentation on **SystemTap**, as well as numerous examples of **SystemTap** scripts.

See Also

- Section B.5, “Changes in SystemTap” provides a comprehensive list of features and improvements over the version of **SystemTap** distributed in the previous release of Red Hat Developer Toolset.

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.

- Chapter 12, *Valgrind* explains how to use **Valgrind** to profile applications and detect memory errors and memory management problems, such as the use of uninitialized memory, improper allocation and freeing of memory, and the use of improper arguments in system calls.

- Chapter 13, *OProfile* explains how to use **OProfile** to determine which sections of code consume the greatest amount of CPU time and why.

- Chapter 14, *Dyninst* documents how to use the Dyninst library to instrument a user-space executable.

CHAPTER 12. VALGRIND

Valgrind is an instrumentation framework that ships with a number of tools for profiling applications. It can be used to detect various memory errors and memory-management problems, such as the use of uninitialized memory or an improper allocation and freeing of memory, or to identify the use of improper arguments in system calls. For a complete list of profiling tools that are distributed with the Red Hat Developer Toolset version of **Valgrind**, see Table 12.1, "Tools Distributed with Valgrind for Red Hat Developer Toolset".

Valgrind profiles an application by rewriting it and instrumenting the rewritten binary. This allows you to profile your application without the need to recompile it, but it also makes **Valgrind** significantly slower than other profilers, especially when performing extremely detailed runs. It is therefore not suited to debugging time-specific issues, or kernel-space debugging.

Red Hat Developer Toolset is distributed with **Valgrind 3.11.0**. This version is more recent than the version included in the previous release of Red Hat Developer Toolset and provides numerous bug fixes and enhancements.

Table 12.1. Tools Distributed with Valgrind for Red Hat Developer Toolset

Name	Description
Memcheck	Detects memory management problems by intercepting system calls and checking all read and write operations.
Cachegrind	Identifies the sources of cache misses by simulating the level 1 instruction cache (I1), level 1 data cache (D1), and unified level 2 cache (L2).
Callgrind	Generates a call graph representing the function call history.
Helgrind	Detects synchronization errors in multithreaded C, C++, and Fortran programs that use POSIX threading primitives.
DRD	Detects errors in multithreaded C and C++ programs that use POSIX threading primitives or any other threading concepts that are built on top of these POSIX threading primitives.
Massif	Monitors heap and stack usage.

12.1. INSTALLING VALGRIND

In Red Hat Developer Toolset, **Valgrind** is provided by the devtoolset-4-valgrind package and is automatically installed with devtoolset-4-perftools. If you intend to use **Valgrind** to profile parallel programs that use the Message Passing Interface (MPI) protocol, also install the devtoolset-4-valgrind-openmpi package by typing the following at a shell prompt as `root`:

```
yum install devtoolset-4-valgrind-openmpi
```

For detailed instructions on how to install Red Hat Developer Toolset and related packages to your system, see Section 1.5, "Installing Red Hat Developer Toolset".

NOTE

Note that if you use **Valgrind** in combination with the **GNU Debugger**, it is recommended that you use the version of **GDB** that is included in Red Hat Developer Toolset to ensure that all features are fully supported.

12.2. USING VALGRIND

To run any of the **Valgrind** tools on a program you want to profile, type the following at a shell prompt:

```
scl enable devtoolset-4 'valgrind [--tool=tool] program [argument...]'
```

See Table 12.1, “Tools Distributed with Valgrind for Red Hat Developer Toolset” for a list of tools that are distributed with **Valgrind**. The argument of the `--tool` command line option must be specified in lower case, and if this option is omitted, **Valgrind** uses **Memcheck** by default. For example, to run **Cachegrind** on a program to identify the sources of cache misses, type:

```
scl enable devtoolset-4 'valgrind --tool=cachegrind program [argument...]'
```

Note that you can execute any command using the `scl` utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset **Valgrind** as default:

```
scl enable devtoolset-4 'bash'
```

IMPORTANT

Red Hat Developer Toolset 4.0 as well as Red Hat Enterprise Linux 7.0 and 7.1 support only the **Open MPI** application binary interface (ABI) version 1.6, whereas Red Hat Enterprise Linux 7.2 supports **Open MPI 1.10**. The two versions are binary incompatible. As a consequence, programs that are built against **Open MPI 1.10** cannot be run under **Valgrind** included in Red Hat Developer Toolset. To work around this problem, use the Red Hat Enterprise Linux 7.2 version of **Valgrind** for programs linked against **Open MPI** version 1.10.

NOTE

To verify the version of Valgrind you are using at any point, type the following at a shell prompt:

```
which valgrind
```

Red Hat Developer Toolset's `valgrind` executable path will begin with `/opt`. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset **Valgrind**:

```
valgrind --version
```

12.3. REBUILDING VALGRIND

The source RPM package for **Valgrind** (`devtoolset-4-valgrind.src.rpm`) requires the openmpi-devel package version 1.3.3 or later. On Red Hat Enterprise Linux 6.8, running the **`yum -y install openmpi-devel`** command results in installing the openmpi-1.10-devel package, and thus the requirement is unsatisfied. As a consequence, **`devtoolset-4-valgrind.src.rpm`** cannot be rebuilt on Red Hat Enterprise Linux 6.8. Note that this problem does not occur in earlier releases of Red Hat Enterprise Linux 6.

12.4. ADDITIONAL RESOURCES

A detailed description of **Valgrind** and its features is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

- valgrind(1) — The manual page for the **`valgrind`** utility provides detailed information on how to use Valgrind. To display the manual page for the version included in Red Hat Developer Toolset, type:

  ```
  scl enable devtoolset-4 'man valgrind'
  ```

- *Valgrind Documentation* — HTML documentation for **Valgrind** is located at **`/opt/rh/devtoolset-4/root/usr/share/doc/devtoolset-4-valgrind-3.9.0/html/index.html`**.

Online Documentation

- Red Hat Enterprise Linux 6 Developer Guide and Red Hat Enterprise Linux 7 Developer Guide — The *Developer Guides* for Red Hat Enterprise Linux 6 and 7 provide more information about **Valgrind** and its **Eclipse** plug-in.

- Red Hat Enterprise Linux 6 Performance Tuning Guide Red Hat Enterprise Linux 7 Performance Tuning Guide — The *Performance Tuning Guides* for Red Hat Enterprise Linux 6 and 7 provide more detailed information about using **Valgrind** to profile applications.

See Also

- Section B.7, “Changes in Valgrind” provides a comprehensive list of features and improvements over the version of **Valgrind** distributed in the previous release of Red Hat Developer Toolset.

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.

- Chapter 10, *memstomp* documents how to use the **memstomp** utility to identify calls to library functions with overlapping memory regions that are not allowed by various standards.

- Chapter 11, *SystemTap* provides an introduction to **SystemTap** and explains how to use it to monitor the activities of a running system.

- Chapter 13, *OProfile* explains how to use **OProfile** to determine which sections of code consume the greatest amount of CPU time and why.

- Chapter 14, *Dyninst* documents how to use the Dyninst library to instrument a user-space executable.

CHAPTER 13. OPROFILE

OProfile is a low overhead, system-wide profiler that uses the performance-monitoring hardware on the processor to retrieve information about the kernel and executables on the system, such as when memory is referenced, the number of level 2 cache (L2) requests, and the number of hardware interrupts received. It consists of a configuration utility, a daemon for collecting data, and a number of tools that can be used to transform the data into a human-readable form. For a complete list of tools that are distributed with the Red Hat Developer Toolset version of **OProfile**, see Table 13.1, “Tools Distributed with OProfile for Red Hat Developer Toolset”.

OProfile profiles an application without adding any instrumentation by recording the details of every nth event. This allows it to consume fewer resources than **Valgrind**, but it also causes its samples to be less precise. Unlike **Valgrind**, which only collects data for a single process and its children in user-space, **OProfile** is well suited to collect system-wide data on both user-space and kernel-space processes, and requires `root` privileges to run.

Red Hat Developer Toolset is distributed with **OProfile 1.1.0**.

Table 13.1. Tools Distributed with OProfile for Red Hat Developer Toolset

Name	Description
`oprofiled`	The **OProfile** daemon that collects profiling data.
`operf`	Intended to replace the deprecated `opcontrol` tool. The `operf` tool uses the Linux Performance Events subsystem, allowing users to target their profiling more precisely, as a single process or system-wide, and allowing **OProfile** to co-exist better with other tools using the performance monitoring hardware on your system. Unlike `opcontrol`, no initial setup is required, and it can be used without the root privileges unless the `--system-wide` option is in use.
`opannotate`	Generates an annotated source file or assembly listing from the profiling data.
`oparchive`	Generates a directory containing executable, debug, and sample files.
`opgprof`	Generates a summary of a profiling session in a format compatible with `gprof`.
`ophelp`	Displays a list of available events.
`opimport`	Converts a sample database file from a foreign binary format to the native format.
`opjitconv`	Converts a just-in-time (JIT) dump file to the Executable and Linkable Format (ELF).
`opreport`	Generates image and symbol summaries of a profiling session.
`ocount`	A new tool for counting the number of times particular events occur during the duration of a monitored command.

13.1. INSTALLING OPROFILE

In Red Hat Developer Toolset, **OProfile** is provided by the devtoolset-4-oprofile package and is automatically installed with devtoolset-4-perftools as described in Section 1.5, “Installing Red Hat Developer Toolset”.

13.2. USING OPROFILE

To run any of the tools that are distributed with **OProfile**, type the following at a shell prompt as **root**:

```
scl enable devtoolset-4 'tool [option...]'
```

See Table 13.1, “Tools Distributed with OProfile for Red Hat Developer Toolset” for a list of tools that are distributed with **OProfile**. For example, to use the **ophelp** command to list available events in the XML format, type:

```
scl enable devtoolset-4 'ophelp -X'
```

Note that you can execute any command using the **scl** utility, causing it to be run with the Red Hat Developer Toolset binaries used in preference to the Red Hat Enterprise Linux system equivalent. This allows you to run a shell session with Red Hat Developer Toolset **OProfile** as default:

```
scl enable devtoolset-4 'bash'
```

NOTE

To verify the version of **OProfile** you are using at any point, type the following at a shell prompt:

```
which operf
```

Red Hat Developer Toolset's **operf** executable path will begin with **/opt**. Alternatively, you can use the following command to confirm that the version number matches that for Red Hat Developer Toolset **OProfile**:

```
operf --version
```

13.3. ADDITIONAL RESOURCES

A detailed description of **OProfile** and its features is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

- oprofile(1) — The manual page named **oprofile** provides an overview of **OProfile** and available tools. To display the manual page for the version included in Red Hat Developer Toolset, type:

  ```
  scl enable devtoolset-4 'man oprofile'
  ```

- opannotate(1), oparchive(1), operf(1), opgprof(1), ophelp(1), opimport(1), opreport(1) — Manual pages for various tools distributed with **OProfile** provide more information on their respective usage. To display the manual page for the version included in Red Hat Developer Toolset, type:

```
scl enable devtoolset-4 'man tool'
```

Online Documentation

- Red Hat Enterprise Linux 6 Developer Guide and Red Hat Enterprise Linux 7 Developer Guide – The *Developer Guides* for Red Hat Enterprise Linux 6 and 7 provide more information on **OProfile**.
- Red Hat Enterprise Linux 6 Deployment Guide — The *Deployment Guide* for Red Hat Enterprise Linux 6 describes in detail how to install, configure, and start using OProfile on this system.
- Red Hat Enterprise Linux 7 System Administrator's Guide — The *System Administrator's Guide* for Red Hat Enterprise Linux 7 documents how to use the `operf` tool.

See Also

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.
- Chapter 11, *SystemTap* provides an introduction to **SystemTap** and explains how to use it to monitor the activities of a running system.
- Chapter 12, *Valgrind* explains how to use **Valgrind** to profile applications and detect memory errors and memory management problems, such as the use of uninitialized memory, improper allocation and freeing of memory, and the use of improper arguments in system calls.
- Chapter 14, *Dyninst* documents how to use the Dyninst library to instrument a user-space executable.

CHAPTER 14. DYNINST

The **Dyninst** library provides an *application programming interface* (API) for instrumenting and working with user-space executables during their execution. It can be used to insert code into a running program, change certain subroutine calls, or even remove them from the program. It serves as a valuable debugging and performance-monitoring tool. The **Dyninst** API is also commonly used along with **SystemTap** to allow non-`root` users to instrument user-space executables.

Red Hat Developer Toolset is distributed with **Dyninst 9.1.0**.

14.1. INSTALLING DYNINST

In Red Hat Developer Toolset, the **Dyninst** library is provided by the devtoolset-4-dyninst package and is automatically installed with devtoolset-4-perftools as described in Section 1.5, "Installing Red Hat Developer Toolset". In addition, it is recommended that you also install the **GNU Compiler Collection** provided by the devtoolset-4-toolchain package.

If you intend to write a custom instrumentation for binaries, install the relevant header files by running the following command as `root`:

```
yum install devtoolset-4-dyninst-devel
```

You can also install API documentation for this library by typing the following at a shell prompt as `root`:

```
yum install devtoolset-4-dyninst-doc
```

For a complete list of documents that are included in the devtoolset-4-dyninst-doc package, see Section 14.3, "Additional Resources". For detailed instructions on how to install optional packages to your system, see Section 1.5, "Installing Red Hat Developer Toolset".

14.2. USING DYNINST

14.2.1. Using Dyninst with SystemTap

To use **Dyninst** along with **SystemTap** to allow non-`root` users to instrument user-space executables, run the `stap` command with the `--dyninst` (or `--runtime=dyninst`) command line option. This tells `stap` to translate a **SystemTap** script into C code that uses the **Dyninst** library, compile this C code into a shared library, and then load the shared library and run the script. Note that when executed like this, the `stap` command also requires the `-c` or `-x` command line option to be specified.

To use the **Dyninst** runtime to instrument an executable file, type the following at a shell prompt:

```
scl enable devtoolset-4 "stap --dyninst -c 'command' [option...]
[argument...]"
```

Similarly, to use the **Dyninst** runtime to instrument a user's process, type:

```
scl enable devtoolset-4 "stap --dyninst -x process_id [option...]
[argument...]"
```

See Chapter 11, *SystemTap* for more information about the Red Hat Developer Toolset version of **SystemTap**. For a general introduction to **SystemTap** and its usage, see the *SystemTap Beginners Guide* for Red Hat Enterprise Linux 6 or the *SystemTap Beginners Guide* for Red Hat Enterprise Linux 7.

Example 14.1. Using Dyninst with SystemTap

Consider a source file named **exercise.C** that has the following contents:

```
#include <stdio.h>

void print_iteration(int value) {
  printf("Iteration number %d\n", value);
}

int main(int argc, char **argv) {
  int i;
  printf("Enter the starting number: ");
  scanf("%d", &i);
  for(; i>0; --i)
    print_iteration(i);
  return 0;
}
```

This program prompts the user to enter a starting number and then counts down to 1, calling the **print_iteration()** function for each iteration in order to print the number to the standard output. To compile this program on the command line using the **g++** compiler from Red Hat Developer Toolset, type the following at a shell prompt:

```
~]$ scl enable devtoolset-4 'g++ -g -o exercise exercise.C'
```

Now consider another source file named **count.stp** with the following contents:

```
#!/usr/bin/stap

global count = 0

probe process.function("print_iteration") {
  count++
}

probe end {
  printf("Function executed %d times.\n", count)
}
```

This **SystemTap** script prints the total number of times the **print_iteration()** function was called during the execution of a process. To run this script on the **exercise** binary file, type:

```
~]$ scl enable devtoolset-4 "stap --dyninst -c './exercise' count.stp"
Enter the starting number: 5
Iteration number 5
Iteration number 4
Iteration number 3
Iteration number 2
Iteration number 1
Function executed 5 times.
```

14.2.2. Using Dyninst as a Stand-alone Application

Before using the **Dyninst** library as a stand-alone application, set the value of the **DYNINSTAPI_RT_LIB** environment variable to the path to the runtime library file. You can do so by typing the following at a shell prompt:

```
export DYNINSTAPI_RT_LIB=/opt/rh/devtoolset-
4/root/usr/lib64/dyninst/libdyninstAPI_RT.so
```

This sets the **DYNINSTAPI_RT_LIB** environment variable in the current shell session.

Example 14.2, "Using Dyninst as a Stand-alone Application" illustrates how to write and build a program to monitor the execution of a user-space process. For a detailed explanation of how to use **Dyninst**, see the resources listed in Section 14.3, "Additional Resources".

Example 14.2. Using Dyninst as a Stand-alone Application

Consider the **exercise.C** source file from Example 14.1, "Using Dyninst with SystemTap": this program prompts the user to enter a starting number and then counts down to 1, calling the **print_iteration()** function for each iteration in order to print the number to standard output.

Now consider another source file named **count.C** with the following contents:

```
#include <stdio.h>
#include <fcntl.h>
#include "BPatch.h"
#include "BPatch_process.h"
#include "BPatch_function.h"
#include "BPatch_Vector.h"
#include "BPatch_thread.h"
#include "BPatch_point.h"

void usage() {
  fprintf(stderr, "Usage: count <process_id> <function>\n");
}

// Global information for counter
BPatch_variableExpr *counter = NULL;

void createCounter(BPatch_process *app, BPatch_image *appImage) {
  int zero = 0;
  counter = app->malloc(*appImage->findType("int"));
  counter->writeValue(&zero);
}

bool interceptfunc(BPatch_process *app,
                   BPatch_image *appImage,
                   char *funcName) {
  BPatch_Vector<BPatch_function *> func;
```

```
    appImage->findFunction(funcName, func);
    if(func.size() == 0) {
      fprintf(stderr, "Unable to find function to instrument()\n");
      exit (-1);
    }
    BPatch_Vector<BPatch_snippet *> incCount;
    BPatch_Vector<BPatch_point *> *points;
    points = func[0]->findPoint(BPatch_entry);
    if ((*points).size() == 0) {
      exit (-1);
    }

    BPatch_arithExpr counterPlusOne(BPatch_plus, *counter,
  BPatch_constExpr(1));
    BPatch_arithExpr addCounter(BPatch_assign, *counter, counterPlusOne);

    return app->insertSnippet(addCounter, *points);
  }


  void printCount(BPatch_thread *thread, BPatch_exitType) {
    int val = 0;
    counter->readValue(&val, sizeof(int));
    fprintf(stderr, "Function executed %d times.\n", val);
  }

  int main(int argc, char *argv[]) {
     int pid;
     BPatch bpatch;
     if (argc != 3) {
       usage();
       exit(1);
     }
     pid = atoi(argv[1]);
     BPatch_process *app = bpatch.processAttach(NULL, pid);
     if (!app) exit (-1);
     BPatch_image *appImage = app->getImage();
     createCounter(app, appImage);
     fprintf(stderr, "Finding function %s(): ", argv[2]);
     BPatch_Vector<BPatch_function*> countFuncs;
     fprintf(stderr, "OK\nInstrumenting function %s(): ", argv[2]);
     interceptfunc(app, appImage, argv[2]);
     bpatch.registerExitCallback(printCount);
     fprintf(stderr, "OK\nWaiting for process %d to exit...\n", pid);
     app->continueExecution();
     while (!app->isTerminated())
       bpatch.waitForStatusChange();
     return 0;
  }
```

Note that a client application is expected to destroy all **`Bpatch`** objects before any of the **Dyninst** library destructors are called. Otherwise the mutator might terminate unexpectedly with a segmentation fault. To work around this problem, set the **`BPatch`** object of the mutator as a local variable in the **`main()`** function. Or, if you need to use **`BPatch`** as a global variable, manually detach all the mutatee processes before the mutator exits.

This program accepts a process ID and a function name as command line arguments and then prints the total number of times the function was called during the execution of the process. You can use the following **Makefile** to build these two files:

```
DTS      = /opt/rh/devtoolset-4/root
CXXFLAGS = -g -I$(DTS)/usr/include/dyninst
LBITS   := $(shell getconf LONG_BIT)

ifeq ($(LBITS),64)
  DYNINSTLIBS = $(DTS)/usr/lib64/dyninst
else
  DYNINSTLIBS = $(DTS)/usr/lib/dyninst
endif

.PHONY: all
all: count exercise

count: count.C
	g++ $(CXXFLAGS) count.C -I /usr/include/dyninst -c
	g++ $(CXXFLAGS) count.o -L $(DYNINSTLIBS) -ldyninstAPI -o count

exercise: exercise.C
	g++ $(CXXFLAGS) exercise.C -o exercise

.PHONY: clean
clean:
	rm -rf *~ *.o count exercise
```

To compile the two programs on the command line using the **g++** compiler from Red Hat Developer Toolset, run the **make** utility as follows:

```
~]$ scl enable devtoolset-4 make
g++ -g -I/opt/rh/devtoolset-4/root/usr/include/dyninst count.C -c
g++ -g -I/opt/rh/devtoolset-4/root/usr/include/dyninst count.o -L
/opt/rh/devtoolset-4/root/usr/lib64/dyninst -ldyninstAPI -o count
g++ -g -I/opt/rh/devtoolset-4/root/usr/include/dyninst exercise.C -o
exercise
```

This creates new binary files called **exercise** and **count** in the current working directory.

In one shell session, execute the **exercise** binary file as follows and wait for it to prompt you to enter the starting number:

```
~]$ ./exercise
Enter the starting number:
```

Do not enter this number. Instead, start another shell session and type the following at its prompt to set the **DYNINSTAPI_RT_LIB** environment variable and execute the **count** binary file:

```
~]$ export DYNINSTAPI_RT_LIB=/opt/rh/devtoolset-
4/root/usr/lib64/dyninst/libdyninstAPI_RT.so
~]$ ./count `pidof exercise` print_iteration
```

```
Finding function print_iteration(): OK
Instrumenting function print_iteration(): OK
Waiting for process 8607 to exit...
```

Now switch back to the first shell session and enter the starting number as requested by the **exercise** program. For example:

```
Enter the starting number: 5
Iteration number 5
Iteration number 4
Iteration number 3
Iteration number 2
Iteration number 1
```

When the **exercise** program terminates, the **count** program displays the number of times the **print_iteration()** function was executed:

```
Function executed 5 times.
```

14.3. ADDITIONAL RESOURCES

A detailed description of Dyninst and its features is beyond the scope of this book. For more information, see the resources listed below.

Installed Documentation

The devtoolset-4-dyninst-doc package installs the following documents in the **/opt/rh/devtoolset-4/root/usr/share/doc/devtoolset-4-dyninst-doc-8.2.1/** directory:

- *Dyninst Programmer's Guide*— A detailed description of the Dyninst API is stored in the **DyninstAPI.pdf** file.
- *DynC API Programmer's Guide*— An introduction to DynC API is stored in the **dynC_API.pdf** file.
- *ParseAPI Programmer's Guide*— An introduction to the ParseAPI is stored in the **ParseAPI.pdf** file.
- *PatchAPI Programmer's Guide*— An introduction to PatchAPI is stored in the **PatchAPI.pdf** file.
- *ProcControlAPI Programmer's Guide*— A detailed description of ProcControlAPI is stored in the **ProcControlAPI.pdf** file.
- *StackwalkerAPI Programmer's Guide*— A detailed description of StackwalkerAPI is stored in the **stackwalker.pdf** file.
- *SymtabAPI Programmer's Guide*— An introduction to SymtabAPI is stored in the **SymtabAPI.pdf** file.
- *InstructionAPI Reference Manual*— A detailed description of the InstructionAPI is stored in the **InstructionAPI.pdf** file.

For information on how to install this package on your system, see Section 14.1, “Installing Dyninst”.

Online Documentation

- Dyninst Home Page — The project home page provides links to additional documentation and related publications.
- Red Hat Enterprise Linux 6 SystemTap Beginners Guide — The *SystemTap Beginners Guide* for Red Hat Enterprise Linux 6 provides an introduction to SystemTap and its usage.
- Red Hat Enterprise Linux 7 SystemTap Beginners Guide — The *SystemTap Beginners Guide* for Red Hat Enterprise Linux 7 provides an introduction to SystemTap and its usage.
- Red Hat Enterprise Linux 6 SystemTap Tapset Reference — The *SystemTap Tapset Reference* for Red Hat Enterprise Linux 6 provides further details about SystemTap.
- Red Hat Enterprise Linux 7 SystemTap Tapset Reference — The *SystemTap Tapset Reference* for Red Hat Enterprise Linux 7 provides further details about SystemTap.

See Also

- Chapter 1, *Red Hat Developer Toolset* provides an overview of Red Hat Developer Toolset and more information on how to install it on your system.
- Chapter 11, *SystemTap* provides an introduction to **SystemTap** and explains how to use it to monitor the activities of a running system.
- Chapter 12, *Valgrind* explains how to use **Valgrind** to profile applications and detect memory errors and memory management problems, such as the use of uninitialized memory, improper allocation and freeing of memory, and the use of improper arguments in system calls.
- Chapter 13, *OProfile* explains how to use **OProfile** to determine which sections of code consume the greatest amount of CPU time and why.

PART VI. GETTING HELP

CHAPTER 15. ACCESSING RED HAT PRODUCT DOCUMENTATION

Red Hat Product Documentation located at https://access.redhat.com/site/documentation/ serves as a central source of information. It is currently translated in 23 languages, and for each product, it provides different kinds of books from release and technical notes to installation, user, and reference guides in HTML, PDF, and EPUB formats.

Below is a brief list of documents that are directly or indirectly relevant to this book.

RED HAT DEVELOPER TOOLSET

- Red Hat Developer Toolset 4.1 Release Notes – The *Release Notes* for Red Hat Developer Toolset 4.1 contain more information.

- Red Hat Software Collections Packaging Guide – The *Software Collections Packaging Guide* explains the concept of Software Collections and documents how to create, build, and extend them.

RED HAT ENTERPRISE LINUX

- Red Hat Enterprise Linux 6 Developer Guide and Red Hat Enterprise Linux 7 Developer Guide – The *Developer Guides* for Red Hat Enterprise Linux 6 and 7 provide more information about libraries and runtime support, compiling and building, debugging, and profiling.

- Red Hat Enterprise Linux 6 Installation Guide – The *Installation Guide* for Red Hat Enterprise Linux 6 explains how to obtain, install, and update the system.

- Red Hat Enterprise Linux 6 Installation Guide and Red Hat Enterprise Linux 7 Installation Guide – The *Installation Guides* for Red Hat Enterprise Linux 6 an 7 explain how to obtain, install, and update the system.

- Red Hat Enterprise Linux 6 Deployment Guide – The *Deployment Guide* for Red Hat Enterprise Linux 6 documents relevant information regarding the deployment, configuration, and administration of Red Hat Enterprise Linux 6.

- Red Hat Enterprise Linux 7 System Administrator's Guide – The *System Administrator's Guide* for Red Hat Enterprise Linux 7 documents relevant information regarding the deployment, configuration, and administration of Red Hat Enterprise Linux 7.

CHAPTER 16. CONTACTING GLOBAL SUPPORT SERVICES

Unless you have a Self-Support subscription, when both the Red Hat Documentation website and Customer Portal fail to provide the answers to your questions, you can contact **Global Support Services** (**GSS**).

16.1. GATHERING REQUIRED INFORMATION

Several items of information should be gathered before contacting GSS.

Background Information

Ensure you have the following background information at hand before calling GSS:

- Hardware type, make, and model on which the product runs
- Software version
- Latest upgrades
- Any recent changes to the system
- An explanation of the problem and the symptoms
- Any messages or significant information about the issue

NOTE

If you ever forget your Red Hat login information, it can be recovered at https://access.redhat.com/site/help/LoginAssistance.html.

Diagnostics

The diagnostics report for Red Hat Enterprise Linux is required as well. This report is also known as a *sosreport* and the program to create the report is provided by the sos package. To install the sos package and all its dependencies on your system, type the following at a shell prompt as **root**:

```
yum install sos
```

To generate the report, run as **root**:

```
sosreport
```

For more information, access the Knowledgebase article at https://access.redhat.com/kb/docs/DOC-3593.

Account and Contact Information

In order to help you, GSS requires your account information to customize their support, as well contact information to get back to you. When you contact GSS ensure you have your:

- Red Hat customer number or Red Hat Network (RHN) login name
- Company name
- Contact name

- Preferred method of contact (phone or email) and contact information (phone number or email address)

Issue Severity

Determining an issue's severity is important to allow the GSS team to prioritize their work. There are four levels of severity.

Severity 1 (urgent)

A problem that severely impacts your use of the software for production purposes. It halts your business operations and has no procedural workaround.

Severity 2 (high)

A problem where the software is functioning, but production is severely reduced. It causes a high impact to business operations, and no workaround exists.

Severity 3 (medium)

A problem that involves partial, non-critical loss of the use of the software. There is a medium to low impact on your business, and business continues to function by utilizing a workaround.

Severity 4 (low)

A general usage question, report of a documentation error, or a recommendation for a future product improvement.

For more information on determining the severity level of an issue, see https://access.redhat.com/support/policy/severity.

Once the issue severity has been determined, submit a service request through the Customer Portal under the **Connect** option, or at https://access.redhat.com/support/contact/technicalSupport.html. Note that you need your Red Hat login details in order to submit service requests.

If the severity is level 1 or 2, then follow up your service request with a phone call. Contact information and business hours are found at https://access.redhat.com/support/contact/technicalSupport.html.

If you have a premium subscription, then after hours support is available for Severity 1 and 2 cases.

Turn-around rates for both premium subscriptions and standard subscription can be found at https://access.redhat.com/support/offerings/production/sla.html.

16.2. ESCALATING AN ISSUE

If you feel an issue is not being handled correctly or adequately, you can escalate it. There are two types of escalations:

Technical escalation

If an issue is not being resolved appropriately or if you need a more senior resource to attend to it.

Management escalation

If the issue has become more severe or you believe it requires a higher priority.

More information on escalation, including contacts, is available at https://access.redhat.com/support/policy/mgt_escalation.html.

16.3. RE-OPENING A SERVICE REQUEST

If there is more relevant information regarding a closed service request (such as the problem reoccurring), you can re-open the request via the Red Hat Customer Portal at https://access.redhat.com/support/policy/mgt_escalation.html or by calling your local support center, the details of which can be found at https://access.redhat.com/support/contact/technicalSupport.html.

IMPORTANT

In order to re-open a service request, you need the original service-request number.

16.4. ADDITIONAL RESOURCES

For more information, see the resources listed below.

Online Documentation

- Getting Started — The *Getting Started* page serves as a starting point for people who purchased a Red Hat subscription and offers the *Red Hat Welcome Kit* and the *Quick Guide to Red Hat Support* for download.

- How can a RHEL Self-Support subscription be used? — A Knowledgebase article for customers with a Self-Support subscription.

- Red Hat Global Support Services and public mailing lists — A Knowledgebase article that answers frequent questions about public Red Hat mailing lists.

APPENDIX A. CHANGES IN VERSION 4.0

The sections below document features and compatibility changes introduced in Red Hat Developer Toolset 4.0.

A.1. CHANGES IN ECLIPSE

Red Hat Developer Toolset 4.0 is distributed with **Eclipse 4.5.0** and other plugins from the Mars release train SR2 (Service Release 2), which provides a number of bug fixes and feature enhancements over the version included in Red Hat Enterprise Linux and the previous version of Red Hat Developer Toolset.

This section contains a comprehensive list of new features and compatibility changes in this release. For details on how to use these new features, refer to the built-in **Eclipse** documentation.

A.1.1. Changes Since Red Hat Developer Toolset 3.1

- A new component, **Eclipse Pydev**, has been added. A full-featured Python IDE with features including code completion, type hinting, code analysis, refactoring, debugging, interactive console, unit test and code coverage integration, and many more.
- A new component, **Eclipse PTP**, has been added. It is a subset of the Eclipse PTP project (http://www.eclipse.org/ptp/) providing support for synchronized projects. Synchronized projects consist of files that are mirrored on the local system as well as on one or more remote systems. Editing occurs locally, and each file is synchronized with the currently active remote system when it is changed, created, or deleted. This enables faster interaction with the files and the editor, more CDT editor features because the files are local, and continued interaction for editing and other functions if the network connection is lost.
- The **Eclipse Platform** has been updated from version 4.4.1 to 4.5.0. As this is a feature release it contains a number of new features, bug fixes and optimizations including, among others, the following:
  - The SWT GTK+3 back end (used by default on Red Hat Enterprise Linux 7) has been significantly enhanced. It is now the default whenever **GTK+ 3.x** is installed on the system.
  - SWT now supports transparency.
  - User experience on Hi-DPI monitors has been improved.
  - The internal **Jetty** server has been updated to **Jetty 9.x** which implements the Servlet 3.1 specification.
  - The Dark theme has been improved and graduated to a supported version (it was a technology preview in the previous version of Red Hat Developer Toolset). It can be accessed on the **General** → **Appearance** preference page.
  - Search speed has been improved. Searching is now up to three to four times faster in case of full workspace searches on multi-core machines.
  - Editor tabs can now be managed. For example, focus can be set to left or right of a closed tab, etc.
  - Hierarchical project view has been added for easier navigation.

  - Search functionality has been added to or enhanced in a number of dialogs, such as **`Open with...`**, **`Open Resource`**, and **`Plug-in Selection`**.
  - The **`Console`** view has been improved. It now supports the **Terminate All** command, as well as output wrapping, scroll lock, and output limit.
  - Refactorings have been added for better code by using lambdas.
  - The speed of the Java compiler (**ecj**) has been improved significantly on generics-heavy code.
  - Support for Java annotation has been enhanced: support for external annotations has been added, as well as the **Annotate** command for adding nullable information in the source. Annotations can now be rendered in the **`Javadoc`** view.
- **Eclipse CDT** (C and C++ Development Tooling) has been updated from version 8.6 to 8.7. This release includes a number of enhancements, including the following:
  - Support for docker-formatted container images has been added. The new subcomponent provides a new, optional feature in CDT, which allows for running and debugging C and C++ applications in docker-formatted container images. To use this feature, select the **Run as C/C++ Container Application** or **Debug as C/C++ Container Application** menu items.
  - Support for project-less execution has been added. Any C or C++ program can be run using the **`Run Configurations`** dialog window without being a part of a project.
  - A wizard has been added for importing existing **Autotools** projects.
- The **Mylyn** task-management subsystem has been updated from version 3.14 to version 3.16. This new release includes the following changes:
  - The Bugzilla connector has been improved to support for Bugzilla versions and fields.
  - Support for saving and restoring breakpoints in task context has been added.
- The **Eclipse Linux Tools** plug-in collection has been updated from version 3.2 to 4.0. This major release includes significant changes, such as:
  - Docker tooling has been added for managing and running docker-formatted container images from inside **Eclipse**.
- **EGit**, a **Git** integration plug-in for **Eclipse**, and **JGit**, a Java library implementing **Git**, have been updated from version 3.6.1 to 4.0.1. This update includes:
  - Support has been added for honoring the **`.gitattributes`** configuration file.
  - Support has been added for pre-commit hooks.
  - Support has been added for the Git-Flow workflow.
  - Usability has been improved significantly. There are better and more numerous tooltips, busy indicators, etc.
  - The **`Staging`** view now supports submodule repositories.

A.2. CHANGES IN GCC

Red Hat Developer Toolset 4.0 is distributed with **GCC 5.2.1**, which provides a number of bug fixes and new features over the version included in Red Hat Enterprise Linux and the previous version of Red Hat Developer Toolset.

A.2.1. Changes Since Red Hat Developer Toolset 3.1

The following features have been added since the release of **GCC** in Red Hat Developer Toolset 3.1:

- Various inter-procedural optimization improvements, including:
  - A new *Identical Code Folding* feature has been added, which tries to merge identical functions in order to save space.
  - Devirtualization optimization has been significantly improved.
  - Better optimization of dead code for C++ inline functions and virtual tables.
  - Better optimization of write-only variables.
- Various link-time optimization improvements, including:
  - **GCC** now utilizes *One Definition Rule*-based merging of C++ types, which allows better devirtualization and alias analysis.
  - **GCC** now features lesser memory usage and faster linking speed.
- Various register-allocation improvements, including:
  - A new control-flow sensitive global register rematerialization optimization has been added, which improves performance.
  - The generation of PIC (Position-Independent Code) has been improved by reusing the PIC hard register.
  - A new inter-procedural register allocator has been added, which brings better performance.
- The **`Undefined Behavior Sanitizer`** (ubsan) gained a few new sanitization options. It is now able to, for instance, detect out-of-bounds accesses and detect various misaligned objects. It is also capable of virtual-pointer checking for C++ code.
- Offloading features of the OpenMP 4.0 specification are now supported by the C, C++, and Fortran compilers.
- Several new warnings options have been added, including:
  - **`-Wswitch-bool`**, which warns whenever a switch statement has an index of a boolean type.
  - **`-Wlogical-not-parentheses`**, which warns about logical not used on the left-hand side operand of a comparison.
  - **`-Wsizeof-array-argument`**, which warns when the **`sizeof`** operator is applied to a parameter that has been declared as an array in a function definition.

  - **-Wbool-compare**, which warns about boolean expressions compared with an integer value different from true or false.

- The preprocessor gained support for the **__has_include** and **__has_include_next** macros, which can be used to test the availability of headers, as well as the **__has_attribute** macro used to determine whether a specified attribute exists.

- A new set of built-in functions for arithmetics with overflow checking has been added: **__builtin_add_overflow**, **__builtin_sub_overflow**, and **__builtin_mul_overflow**.

- The default mode for C is now **-std=gnu11** instead of **-std=gnu89**.

- The C++ compiler now supports many C++14 features, for instance: variable templates, aggregates with non-static data member initializers, the extended **constexpr** specifier, sized deallocation functions, and others. The C++ compiler also supports several new warnings, for example, to help developers annotate programs with **final** specifiers.

- The Runtime Library (**libstdc++**) has been greatly improved. Note that unlike the upstream **GCC 5.x**, the Red Hat Developer Toolset 4.1 version of **GCC** does not use the new ABI.

  - **libstdc++** now has full support for the C++11 standard with the exception of the new implementations of the **std::string** and **std::list** classes because the Red Hat Developer Toolset 4.1 version of **GCC** uses the old ABI.

    Other notable features include, for example, movable and swappable **iostream** classes, support for the **std::align** function and the **std::aligned_union** class template, locale facets for Unicode conversion, and atomic operations for the **std::shared_ptr** class template.

  - **libstdc++** now has full experimental support for the C++14 standard. This includes, for example, the **std::is_final** type trait.

  - The experimental support for the TS Library Fundamentals has been improved.

- **GCC** now contains the **libgccjit** library, which allows users to build **GCC** as a shared library for embedding in other processes, suitable for Just-In-Time (JIT) compilation to machine code.

- **GCC** now contains support for new ISA (Instruction Set Architecture) extensions, including AVX512-VL (Vector Length Extensions), AVX512-BW (Byte and Word Instructions), and AVX512-DQ (Doubleword and Quadword Instructions) extensions on top of the already existing AVX-512 extensions.

- **Pointer Bounds Checker** has been added to **GCC**, which serves as a bounds violation detector. This tool only works on 32-bit a 64-bit Intel Linux targets with the new ISA extension Intel MPX support.

- The Intel Memory Protection Extensions (MPX) support has been added to **GCC** in Red Hat Developer Toolset 4.0 for use with future releases of Red Hat Enterprise Linux 7 (beyond Red Hat Enterprise Linux 7.2) when the kernel enablement is complete. This feature will provide a set of extensions to the x86 Instruction Set Architecture (ISA), which can be used for bounds checking when performing pointer accesses. Note that this feature will not be supported on Red Hat Enterprise Linux 6.

A.3. CHANGES IN BINUTILS

Red Hat Developer Toolset 4.0 is distributed with **binutils 2.25**, which provides a number of bug fixes and feature enhancements over the version included in Red Hat Enterprise Linux and the previous version of Red Hat Developer Toolset. Below is a comprehensive list of new features in this release.

A.3.1. Changes Since Red Hat Enterprise Linux 6.7

The following features have been added since the release of **binutils** in Red Hat Enterprise Linux 6.7:

The GNU assembler (**as**), GNU linker (**ld**), and other binary tools that are part of **binutils** are now released under the GNU General Public License, version 3.

A.3.1.1. GNU Linker

Another ELF linker, **gold**, is now available in addition to **ld**, the existing GNU linker. **gold** is intended to be a drop-in replacement for **ld**, so **ld**'s documentation is intended to be the reference documentation. **gold** supports most of **ld**'s features, except notable ones such as MRI-compatible linker scripts, cross-reference reports (**--cref**), and various other minor options. It also provides significantly improved link time with very large C++ applications.

In Red Hat Developer Toolset 4.0, the **gold** linker is not enabled by default. Users can explicitly switch between **ld** and **gold** by using the **alternatives** mechanism.

A.3.1.1.1. New Features

The following features have been added since the release of **binutils** included in Red Hat Enterprise Linux 6.7:

- A new **INPUT_SECTION_FLAGS** keyword has been added to the linker script language. This keyword can be used to select input sections by section header flags.
- A new **SORT_BY_INIT_PRIORITY** keyword has been added to the linker script language. This keyword can be used to sort sections by numerical value of the GCC ***init_priority*** attribute encoded in the section name.
- A new **SORT_NONE** keyword has been added to the linker script language. This keyword can be used to disable section sorting.
- A new linker-provided symbol, **__ehdr_start**, has been added. When producing ELF output, this symbol points to the ELF file header (and nearby program headers) in the program's memory image.

A.3.1.1.2. Compatibility Changes

The following compatibility changes have been made since the release of **binutils** included in Red Hat Enterprise Linux 6.7:

- The **--copy-dt-needed-entries** command line option is no longer enabled by default. Instead, **--no-copy-dt-needed-entries** is now the default option.
- Evaluation of linker script expressions has been significantly improved. Note that this can negatively affect scripts that rely on undocumented behavior of the old expression evaluation.

A.3.1.2. GNU Assembler

A.3.1.2.1. New Features

The following features have been added since the release of **binutils** included in Red Hat Enterprise Linux 6.7:

- The GNU Assembler no longer requires double ampersands in macros.
- A new **`--compress-debug-sections`** command line option has been added to enable compression of DWARF debug information sections in the relocatable output file. Compressed debug sections are currently supported by the **`readelf`**, **`objdump`**, and **`gold`** tools, but not by **`ld`**.
- Support for **`.bundle_align_mode`**, **`.bundle_lock`**, and **`.bundle_unlock`** directives for x86 targets has been added..
- On x86 architectures, the GNU Assembler now allows **`rep bsf`**, **`rep bsr`**, and **`rep ret`** syntax.

A.3.1.3. Other Binary Tools

A.3.1.3.1. New Features

The following features have been added since the release of **binutils** included in Red Hat Enterprise Linux 6.7:

- The **`readelf`** and **`objdump`** tools can now display the contents of the **`.debug.macro`** sections.
- New **`--dwarf-start`** and **`--dwarf-end`** command line options have been added to the **`readelf`** and **`objdump`** tools. These options are used by the new Emacs mode (see the **`dwarf-mode.el`** file).
- A new **`--interleave-width`** command line option has been added to the **`objcopy`** tool to allow the use of the **`--interleave`** to copy a range of bytes from the input to the output.
- A new **`--dyn-syms`** command line option has been added to the **`readelf`** tool. This option can be used to dump dynamic symbol table.
- A new tool, **`elfedit`**, has been added to **binutils**. This tool can be used to directly manipulate ELF format binaries.
- A new command line option **`--addresses`** (or **`-a`** for short) has been added to the **`addr2line`** tool. This option can be used to display addresses before function and source file names.
- A new command line option **`--pretty-print`** (or **`-p`** for short) has been added to the **`addr2line`** tool. This option can be used to produce human-readable output.
- Support for **`dwz -m`** optimized debug information has been added.
- The devtoolset-2-binutils-devel package now provides the **`demangle.h`** header file.

A.3.2. Changes Since Red Hat Enterprise Linux 7.2

The following bugs have been fixed and features added since the release of **binutils** in Red Hat Enterprise Linux 7.2:

- The detection of uncompressed `.debug_str` sections has been fixed.
- The decoding of abbreviations using the `DW_FORM_ref_addr` attribute has been fixed.
- The **objcopy** utility now supports wildcards for section names in command line options.
- The BFD linker script language now supports the `ALIGN_WITH_INPUT` directive for output sections. The directive directs the linker to compute the maximum alignment of the associated input sections and use that alignment for the output section.
- The AVX-512 (512-bit Advanced Vector Extensions) are now supported.

A.3.3. Changes Since Red Hat Developer Toolset 3.1

The following features have been added since the release of **binutils** in Red Hat Developer Toolset 3.1:

A.3.3.1. GNU Linker

- Binaries in the COFF/PE format now once again contain real timestamps by default. This was done so that the binaries will be compatible with tools from other vendors. However, including timestamps in binaries means that two identical builds made at different times will not compare as identical. The timestamps can be disabled using a command-line option, `--no-insert-timestamp`.
- Binaries in the COFF/PE format can now have a build-id added to them using the `--build-id` command line option. This behaves in the same way as the `--build-id` option does for binaries in the ELF format.
- Support has been added for the AVR Tiny microcontrollers and the Andes NDS32 architecture.
- Support for the Openrisc and OR32 architectures has been replaced with support for the OR1K architecture.

A.3.3.2. GNU Assembler

- The ARM assembler now accepts output from the **CodeComposer Studio** tool. This is enabled using a new command line option, `-mccs`.
- Support has been added for the AVR Tiny microcontrollers and the Andes NDS32 architecture.
- Support for the Openrisc and OR32 architectures has been replaced with support for the OR1K architecture.

A.3.3.3. Other Binary Tools

- The default behaviour of the `strings` program has changed. Instead of displaying strings only found in data sections of a program, it will now search all of the program, including the code sections. The old behaviour can be restored using a new command-line option, `--data`.

  The change was made as a security enhancement because scanning the entire program for strings does not require any analysis of the program itself. The analysis code might contain bugs, which could be triggered by specially crafted bogus programs, thus exposing the system itself to attack.

- The **`strings`** utility also has a new command-line option, **`--include-all-whitespace`**. This stops **`strings`** from splitting its output when it encounters a carriage return or the line feed character.

- The **`objcopy`** utility has a new command-line option, **`--dump-section`**. This allows individually named sections to be extracted from a program and copied into another file.

A.4. CHANGES IN ELFUTILS

Red Hat Developer Toolset 4.0 is distributed with **elfutils 0.163**, which provides a number of bug fixes and feature enhancements over the version included in the previous release of Red Hat Developer Toolset. Below is a comprehensive list of new features in this release.

A.4.1. Changes Since Red Hat Developer Toolset 3.1

The following features have been added since the release of **elfutils 0.161** in Red Hat Developer Toolset 3.1:

- The following changes have been introduced in the **libdw** library:
  - A new header file, **`elfutils/known-dwarf.h`**, has neen added.
  - The preliminary DWARF5 constants (**`DW_TAG_atomic_type`**, **`DW_LANG_Fortran03`**, and **`DW_LANG_Fortran08`**) have been added to the **`elfutils/dwarf.h`** header file.
  - The **`dwarf_peel_type ()`** function now also handles the **`DW_TAG_atomic_type`** constant.
- The following changes and improvements have been introduced in the **eu-addr2line** tool:
  - Input addresses are now always interpreted as hexadecimal numbers, never as octal or decimal numbers.
  - A new option, **`-a`** or **`--addresses`**, has been added for printing addresses before each entry.
  - A new option, **`-C`** or **`--demangle`**, has been added for showing demangled symbols.
  - A new option, **`--pretty-print`**, has been added for printing all information on one line.

A.5. CHANGES IN DWZ

Red Hat Developer Toolset 4.0 is distributed with **dwz 0.12**, which provides the following bug fix over the version included in Red Hat Enterprise Linux and the previous version of Red Hat Developer Toolset.

A.5.1. Changes Since Red Hat Developer Toolset 3.1 and Red Hat Enterprise Linux 7.2

The following bug has been fixed since the release of **dwz** in Red Hat Developer Toolset 3.1 and Red Hat Enterprise Linux 7.2:

- The bug which resulted in the possibility of **dwz** producing binaries or shared libraries with unaligned non-allocated ELF sections has been fixed.

A.6. CHANGES IN GDB

Red Hat Developer Toolset 4.0 is distributed with **GDB 7.10**, which provides a number of bug fixes and improvements over the Red Hat Enterprise Linux system version and the version included in the previous release of Red Hat Developer Toolset. Below is a comprehensive list of new features in this release.

A.6.1. Changes Since Red Hat Developer Toolset 3.1

The following features have been added since the release of **GDB** in Red Hat Developer Toolset 3.1:

New Features

- **GDB** now honors the content of the `/proc/PID/coredump_filter` file (`PID` is the process ID). This file can be used to specify the types of memory mappings that will be included in a corefile. For more information, please see the core(5) manual page.

  GDB also has a new command: `set use-coredump-filter on|off`. It allows to set whether **GDB** reads the content of the `/proc/PID/coredump_filter` file when generating a corefile.

- The `info os` command can now be used to display information about CPUs. Run `info os cpus` to list all CPUs and cores on the system.

- The `info source` command can now be used to display the producer string (if it was included in the debugging information). The string usually contains the compiler version and command-line arguments.

- Directory names supplied to the `set sysroot` commands may be prefixed with `target:` to instruct **GDB** to access shared libraries from the target system, regardless of whether it is local or remote. This replaces the `remote:` prefix. The default system root has been changed from "" (empty) to `target:`. For backward compatibility, `remote:` is automatically converted to `target:`.

- The system root specified by the `set sysroot` command is prepended to the filename of the main executable (if reported to **GDB** as absolute by the operating system) when starting processes remotely and when attaching to already-running local or remote processes.

- **GDB** now supports automatic location and retrieval of executable files from remote targets. Remote debugging can now be initiated using only the `target remote` or `target extended-remote` commands (no `set sysroot` or `file` commands are required). See also the section called "New Remote Packets" below.

- Support for the Verilog hexadecimal format has been added to the **dump** command.

- **GDB** and **gdbserver** are now able to access executable and shared library files without the `set sysroot` command when attaching to processes running in different mount namespaces from the debugger. This makes it possible to attach to processes in containers as simply as `gdb -p PID` or `gdbserver --attach PID`. See also the section called "New Remote Packets" below.

- Support for completion for all of the available register groups, including target-specific groups, has been added to the **tui reg** command.

- The size of **GDB**'s command history is no longer determined by reading the **HISTSIZE** environment variable. Instead, the dedicated **GDBHISTSIZE** environment variable is now used. To disable truncation of command history, set **GDBHISTSIZE** to **-1** or leave it empty. Non-numeric values are ignored.

- Support for fork events on extended-remote Linux targets has been added. This makes it possible to use the **follow-fork-mode** and **detach-on-fork** options on targets with Linux kernels 2.5.60 and higher for both the **fork** and **vfork** calls, as well as **fork** and **vfork** catchpoints.

- Using new **gdbfeatures** (**fork-events** and **vfork-events**), the **qSupported** packet allows **GDB** to request support for **fork** and **vfork** events, and the **qSupported** response can contain the corresponding **stubfeatures**. Standard **set** and **show** commands can be used to set and display whether these features are enabled.

- The **info record** command now displays the format of the recording and when the btrace record target is used, it also shows the branch-tracing configuration for the current thread. In case of the BTS format, the command displays the ring buffer size.

- Support for DTrace USDT (Userland Static Defined Tracing) probes has been added.

- **GDB** now supports the compilation and injection of source code into the inferior. **GDB** will use **GCC 5.0** or higher built with **libcc1.so** to compile the source code to object code, and if successful, inject and execute that code within the current context of the inferior. Currently, the C language is supported. The commands used for interfacing with this new feature are:

  ```
  compile code [-raw|-r] [--] [source code]
  ```

  ```
  compile file [-raw|-r] filename
  ```

- The **dll-symbols** command and its two aliases (**add-shared-symbol-files** and **assf**) have been removed. Use the **sharedlibrary** command or its alias **share** instead.

- On resume, **GDB** now always passes the signal the program had stopped for to the thread the signal was sent to, even if the user changed threads before resuming. Previously, **GDB** would often (but not always) deliver the signal to the thread that happened to be current at resume time.

- Conversely, the **signal** command now consistently delivers the requested signal to the current thread. **GDB** now asks for confirmation if the program had stopped for a signal, and the user switched threads meanwhile.

- The **off** and **auto** states of the **breakpoint always-inserted** mode have been merged into one. Now, when the **breakpoint always-inserted** mode is set to **off**, **GDB** does not remove breakpoints from the target until all threads stop, even in non-stop mode. The **auto** mode has been removed, and **off** is now the default mode.

- The **--xdb** command-line option (HP-UX XDB compatibility mode) has been removed.

New Remote Packets
A number of new remote packets have been added. See Table A.1, “New Remote Packets” for a complete list.

Table A.1. New Remote Packets

Remote Packet	Description
`qXfer:btrace-conf:read`	Return the branch-trace configuration for the current thread.
`Qbtrace-conf:bts:size`	Set the requested ring-buffer size for branch tracing in the BTS format.
`swbreak stop` *`reason`*	Indicates a memory breakpoint instruction was executed, irrespective of whether it was **GDB** that planted the breakpoint or the breakpoint was hardcoded in the program. This is required for correct non-stop mode operation.
`hwbreak stop` *`reason`*	Indicates the target stopped for a hardware breakpoint. This is required for correct non-stop mode operation.
`vFile:fstat:`	Return information about files on the remote system.
`qXfer:exec-file:read`	Return the full absolute name of the file that was executed to create a process running on the remote system.
`vFile:setfs:`	Select the file system on which `vFile:` operations with file name arguments will operate. This is required for **GDB** to be able to access files on remote targets where the remote stub does not share a common file system with the inferior(s).
`fork stop` *`reason`*	Indicates that a `fork` system call was executed.
`vfork stop` *`reason`*	Indicates that a `vfork` system call was executed.
`vforkdone stop` *`reason`*	Indicates that a `vfork` child of the specified process has executed an `exec` or `exit` function, allowing the `vfork` parent to resume execution.

Python Scripting Support

Python scripting support has been improved:

- Objects of the `gdb.Objfile` class have a new attribute, *`username`*, which is the user-specified name of the objfile.
- Support for writing frame unwinders in Python has been added.
- A new method, `optimized_out`, has been added to `gdb.Type` objects. It is used for returning the optimized out `gdb.Value` instance of this type.
- New methods, `reference_value` and `const_value`, have been added to `gdb.Value` objects. They are used to return a reference to the value and the `const` version of the value, respectively.

- Auto-loading of Python scripts contained in the **`.debug_gdb_scripts`** special section is now supported.
- **`Xmethods`** can now specify the type of the result.
- You can now access frame registers from Python scripts.
- The **`producer`** attribute for **`gdb.Symtab`** objects has been added.
- The **`progspace`** attribute for **`gdb.Objfile`** objects has been added. It is the **`gdb.Progspace`** object of the containing program space.
- The **`owner`** attribute for **`gdb.Objfile`** objects has been added.
- The **`build_id`** attribute for **`gdb.Objfile`** objects has been added. It is the build ID generated when the file was built.
- The **`add_separate_debug_file`** method has been added to **`gdb.Objfile`** objects.
- A new event, **`gdb.clear_objfiles`**, has been added. It is triggered when a new file for debugging is selected.
- You can now add attributes to **`gdb.Objfile`** and **`gdb.Progspace`** objects.
- A new function, **`gdb.lookup_objfile`**, has been added.

The following new events are triggered when the state of the inferior is modified by **GDB**.

gdb.events.inferior_call_pre
Function call is about to be made.

gdb.events.inferior_call_post
Function call has just been made.

gdb.events.memory_changed
A memory location has been altered.

gdb.events.register_changed
A register has been altered.

The following new Python-based convenience functions have been added.

- **`$_caller_is(name [, number_of_frames])`**
- **`$_caller_matches(regexp [, number_of_frames])`**
- **`$_any_caller_is(name [, number_of_frames])`**
- **`$_any_caller_matches(regexp [, number_of_frames])`**

New Commands
The following new commands have been added:

`set serial parity odd|even|none`, `show serial parity`

Set or show parity for remote serial I/O.

`maint print symbol-cache`

Print the contents of the symbol cache.

`maint print symbol-cache-statistics`

Print statistics of symbol cache usage.

`maint flush-symbol-cache`

Flush the contents of the symbol cache.

`record btrace bts`, `record bts`

Begin branch-trace recording using the Branch Trace Store (BTS) format.

`tui enable`, `tui disable`

Enable or and disable the TUI (Text User Interface) mode.

`set mpx bound`, `show mpx bound`

Investigate bound tables in Intel(R) MPX-enabled applications.

`maint info btrace`

Display information about branch-tracing internals.

`maint btrace packet-history`

Display raw branch-tracing data.

`maint btrace clear-packet-history`

Discard stored raw branch-tracing data.

`maint btrace clear`

Discard all branch-tracing data. The next invocation of the **`record`** command will fetch and process it again.

`demangle [-l` *`language`*`] [--]` *`name`*

Demangle *name* in the specified *language* or the current language if the parameter is omitted. This command replaces the renamed **`maint demangle`** command. The old command name is kept as a no-op command to avoid **`maint demangle`** being interpreted as **`maint demangler-warning`**.

`queue-signal` *`signal-name-or-number`*

Queue a signal to be delivered to the thread when it is resumed.

`maint print user-registers`

Display all currently available user registers.

`compile code [-r|-raw] [--] [source code]`

Compile, inject, and execute in the inferior the executable object code produced by compiling the provided source code.

`compile [file -r|-raw]` ***`filename`***

Compile and inject into the inferior the executable object code produced by compiling the source code stored in the filename provided.

`compile print`

Evaluate an expression using the compiler and display the result.

New Options

The following new options have been added:

`set debug dwarf-die`, `show debug dwarf-die`

Renamed from **`set debug dwarf2-die`** and **`show debug dwarf2-die`** respectively.

`set debug dwarf-read`, `show debug dwarf-read`

Renamed from **`set debug dwarf2-read`** and **`show debug dwarf2-read`** respectively.

`maint set dwarf always-disassemble`, `maint show dwarf always-disassemble`

Renamed from **`maint set dwarf2 always-disassemble`** and **`maint show dwarf2 always-disassemble`** respectively.

`maint set dwarf max-cache-age`, `maint show dwarf max-cache-age`

Renamed from **`maint set dwarf2 max-cache-age`** and **`maint show dwarf2 max-cache-age`** respectively.

`set debug dwarf-line`, `show debug dwarf-line`

Control display of debugging info regarding DWARF line processing.

`set max-completions`, `show max-completions`

Set the maximum number of candidates to be considered during completion. The default value is **`200`**. This limit allows **GDB** to avoid generating large completion lists, the computation of which can cause the debugger to become temporarily unresponsive.

`set history remove-duplicates`, `show history remove-duplicates`

Control the removal of duplicate history entries.

`maint set symbol-cache-size`, `maint show symbol-cache-size`

Control the size of the symbol cache.

`set record btrace bts buffer-size`, `show record btrace bts buffer-size`

Set and show the size of the ring buffer used for branch tracing in the BTS format. The obtained size may differ from the requested size. To see the obtained buffer size, use the **`info record`** command.

`set debug linux-namespaces`, `show debug linux-namespaces`

Control display of debugging info regarding Linux namespaces.

`set debug symbol-lookup`, **`show debug symbol-lookup`**

Control display of debugging info regarding symbol lookup.

The **`thread apply all`** command now supports a new option, **`-ascending`**, for calling its specified command for all threads in an ascending order.

Change in the Machine Interface Interpreter (GDB/MI)
The following change has been made the GDB/MI.

- The **`-list-thread-groups`** command outputs an exit-code field for inferiors that have exited.

A.7. CHANGES IN STRACE

Red Hat Developer Toolset 4.0 is distributed with **strace 4.10**, which provides a number of bug fixes and feature enhancements over the version included in Red Hat Enterprise Linux and the previous version of Red Hat Developer Toolset. Below is a comprehensive list of new features in this release.

A.7.1. Changes Since Red Hat Developer Toolset 3.1 and Red Hat Enterprise Linux 6.7 and 7.2

The following improvements have been added since the release of **strace** in Red Hat Developer Toolset 3.1 and Red Hat Enterprise Linux 6.7 and 7.2:

- An experimental option, **`-k`**, has been added, which prints the execution stack trace of traced processes after each system call.

- A new option, **`-w`**, has been added, which produces statistics on the differences between the start and end of every system call.

- A new option, **`-yy`**, has been added to print protocol and address information associated with socket file descriptors.

- The **`-e read=set`** and **`-e write=set`** options have been extended to cover the following system calls: **`sendmsg`**, **`recvmsg`**, **`sendmmsg`**, and **`recvmmsg`**.

- Full decoding of 64-bit capability sets has been implemented.

- Support for the Bionic and musl standard C libraries has been improved.

- Tracing of x86 personality processes has been improved.

- The decoding of the following system calls has been implemented: **`add_key`**, **`getrandom`**, **`ioprio_get`**, **`ioprio_set`**, **`kexec_load`**, **`keyctl`**, **`renameat2`**, **`request_key`**, and **`seccomp`**.

- The decoding of the **`select`**, **`pselect`**, and **`io_submit`** system calls has been made more robust.

- The decoding of the following system calls has been improved: **`delete_module`**, **`fanotify_init`**, **`fanotify_mark`**, **`fcntl`**, **`getsockopt`**, **`setdomainname`**, **`sethostname`**, **`setns`**, **`setsockopt`**, **`sync_file_range`**, and **`sysinfo`**.

- The socket decoder has been enhanced to support Bluetooth sockets.
- The decoding of the following has been implemented:
    - all prctl commands
    - parametrized ioctl commands
    - evdev ioctl commands
    - v4l ioctl commands
    - **SG_IO** v4 ioctl commands
    - **FIFREEZE**, **FITHAW**, **FITRIM** ioctl commands
    - **FALLOC_FL_*** fallocate flags
    - **rt_sigreturn** signal mask
- The decoding of the following has been improved:
    - signal bitmasks
    - file descriptors
    - the **siginfo_t** data structure
    - the **PF_NETLINK** preprocessor macro
    - unlisted ioctl commands
    - the **cmsghdr** struct
    - the wait status
- The **CAP_***, **CLOCK_***, **PR_***, **PTRACE_***, **SCHED_***, **SO_***, **SOL_***, **SWAP_FLAG_***, and **TFD_*** constants have been updated.
- New system call entries have been added to match **Linux 3.19**.
- Lists of signal constants, errno constants, and ioctl commands have been updated from **Linux 3.19**.
- The test that required a Red Hat Enterprise Linux 7 kernel has been disabled.

A.8. CHANGES IN LTRACE

Red Hat Developer Toolset 4.0 is distributed with **ltrace 0.7.91**. While the base version of **ltrace** remains the same as in the previous release of Red Hat Developer Toolset, various bug fixes have been ported. Below is a list of the most important bug fixes in this release.

A.8.1. Changes Since Red Hat Developer Toolset 3.1 and Red Hat Enterprise Linux 7.2

The following bugs have been fixed since the release of **ltrace** in Red Hat Developer Toolset 3.1 and Red Hat Enterprise Linux 7.2:

- In configuration files, the user may now put a space between an indentifier and a following open parenthesis.
- The **ltrace** test suite now works correctly regardless of any installed `ltrace.conf` configuration file or other environmental effects.

A.9. CHANGES IN SYSTEMTAP

Red Hat Developer Toolset 4.0 is distributed with **SystemTap 2.8**, which provides a number of bug fixes and feature enhancements over the version included in the previous release of Red Hat Developer Toolset. Below is a list of new features in this release.

A.9.1. Changes Since Red Hat Developer Toolset 3.1

The following features have been added since the release of **SystemTap** included in Red Hat Developer Toolset 3.1:

- Documentation has been improved, including manual pages and cross references.
- Diagnostics have been improved. The output is now colorized.
- Namespace-aware identifier lookup functions have been added to the `tapset` library.
- Golang support has been improved.
- String passing has been optimized.
- Functions in the `tapset` library now support address-to-file:line mapping.
- System-call probes in the `tapset` have been improved, especially the non-DWARF `nd_syscall` variants.

NOTE

Incompatibility problems with old scripts can be resolved using the backward-compatibility option, `--compatible` ***`version`***, where *version* is the version of **SystemTap** for which the script was written.

A.10. CHANGES IN OPROFILE

Red Hat Developer Toolset 4.0 is distributed with **OProfile 1.1.0**, which provides a number of bug fixes and feature enhancements over the version included in the previous release of Red Hat Developer Toolset. Below is a list of new features in this release.

A.10.1. Changes Since Red Hat Developer Toolset 3.1

The following features have been added since the release of **OProfile** included in Red Hat Developer Toolset 3.1:

- The deprecated **`opcontrol`** and **`oprof_start`** commands have been removed from **OProfile 1.1.0**. The **`operf`** command should be used in their place. The **`opreport`** command is still used to analyze data stored in the **`oprofile_data/`** directory by **`operf`**.

- Support for new Intel processors, such as 6th Generation Intel Core Processors, Intel Xeon Processor D, and Airmont, has been added.

- Fixes have been included to avoid dropping samples from Java programs using Just-In-Time (JIT) translation due to use of anonymous hugepages and changes in the anonymous page mappings.

APPENDIX B. CHANGES IN VERSION 4.1

The sections below document features and compatibility changes introduced in Red Hat Developer Toolset 4.1.

B.1. CHANGES IN ECLIPSE

Red Hat Developer Toolset 4.1 is distributed with **Eclipse 4.5.2** and other plugins from the Mars release train, which provides a number of bug fixes and feature enhancements over the version included in Red Hat Enterprise Linux and the previous version of Red Hat Developer Toolset.

This section contains a comprehensive list of new features and compatibility changes in this release. For details on how to use these new features, refer to the built-in **Eclipse** documentation.

B.1.1. Changes Since Red Hat Developer Toolset 4.0

- A new component, **Eclipse TM Terminal**, has been added. It is a small, re-usable component for terminal emulation and remote access.
- A new component, **Eclipse Dynamic Languages Toolkit**, has been added. It is a comprehensive Ruby, TCL, and Shell script IDE with features including code running and debugging, auto-completion, outlining, support for executing unit tests and graphically presenting the results, and others.
- The **Eclipse Platform** has been updated from version 4.5.0 to 4.5.2. As this is a bugfix release, it does not contains new features, only bug fixes and optimizations, including the following:
  - The SWT GTK+3 back end (used by default on Red Hat Enterprise Linux 7) has been optimized for performance.
  - SWT has gained better compatibility with GTK+ 3.14 (Red Hat Enterprise Linux 7.2) and features drawing fixes and other improvements.
  - The Java formatter has been significantly improved.
  - The stability and behavior of Drag'n'Drop operations has been improved.
- **Eclipse CDT** (C and C++ Development Tooling) has been updated from version 8.7 to 8.8.1. This release includes a number of enhancements, including the following:
  - **`Memory`** views have been improved, including support for **`Find/Replace`** and **`Add Watchpoint`**.
  - The readability of the **`Dissassembly`** view has been improved.
  - Support for user-defined literals has added.
- The **Mylyn** task-management subsystem has been updated from version 3.16 to 3.18. This new release includes the following changes:
  - The **`Task list`** view has been improved.
  - The saving and restoring of breakpoints in the task context is now supported.
- The **Eclipse Linux Tools** plug-in collection has been updated from version 3.2 to 4.0. This major release includes significant changes, such as:

  - A new subcomponent, the **Vagrant plugin**, has been added. The plugin supports interaction with Vagrant boxes and virtual machines.
  - The **Docker plugin** gained support for searching the Docker registry for images and for running images directly. The plugin now also includes a full-featured interactive shell support.
  - The **OProfile plugin** has gained support for the `ocount` tool.
  - The **RPM plugin** has gained support for weak dependencies (**`Suggests`** and **`Recommends`**).
  - Manual pages are browsable and searchable through the Eclipse Help system.
- **EGit**, a **Git** integration plug-in for **Eclipse**, and **JGit**, a Java library implementing **Git**, have been updated from version 4.0.1 to 4.2.0. This update includes:
  - Basic support for push certificates has been added.
  - Performance has been improved by making use of the Java NIO (Non-blocking I/O).
  - Support for the Remote [Add|List|Remove] Command has been added to the **JGit** API.
  - Support for the gitflow model has been improved.
  - The **`Staging`** view now supports submodule repositories.
- The **Eclipse PyDev** development environment for Python has been updated from version 4.1.0 to 4.5.4. This release includes a number of changes, such as:
  - The debugger has been improved significantly.
  - The **PyDev Package Explorer** has been improved to provide more information when all elements are filtered.
  - Code completion has been improved.
  - A new Python search page has been added, and the search backend is now backed by the Lucene engine.
- The **Eclipse PTP** (Parallel Tools Platform) has been updated from version 4.1.0 to 4.5.4.

B.2. CHANGES IN BINUTILS

Red Hat Developer Toolset 4.1 is distributed with **binutils 2.25.1**, which provides a number of bug fixes and feature enhancements over the version included in Red Hat Enterprise Linux and the previous version of Red Hat Developer Toolset. Below is a comprehensive list of new features in this release.

B.2.1. Changes Since Red Hat Enterprise Linux 6.8

The following features have been added since the release of **binutils** in Red Hat Enterprise Linux 6.8:

The GNU assembler (**`as`**), GNU linker (**`ld`**), and other binary tools that are part of **binutils** are now released under the GNU General Public License, version 3.

B.2.1.1. GNU Linker

Another ELF linker, **`gold`**, is now available in addition to **`ld`**, the existing GNU linker. **`gold`** is intended to be a drop-in replacement for **`ld`**, so **`ld`**'s documentation is intended to be the reference documentation. **`gold`** supports most of **`ld`**'s features, except notable ones such as MRI-compatible linker scripts, cross-reference reports (`--cref`), and various other minor options. It also provides significantly improved link time with very large C++ applications.

In Red Hat Developer Toolset 4.1, the **`gold`** linker is not enabled by default. Users can explicitly switch between **`ld`** and **`gold`** by using the **`alternatives`** mechanism.

B.2.1.1.1. New Features

The following features have been added since the release of **binutils** included in Red Hat Enterprise Linux 6.8:

- A new **`INPUT_SECTION_FLAGS`** keyword has been added to the linker script language. This keyword can be used to select input sections by section header flags.
- A new **`SORT_BY_INIT_PRIORITY`** keyword has been added to the linker script language. This keyword can be used to sort sections by numerical value of the GCC ***`init_priority`*** attribute encoded in the section name.
- A new **`SORT_NONE`** keyword has been added to the linker script language. This keyword can be used to disable section sorting.
- A new linker-provided symbol, **`__ehdr_start`**, has been added. When producing ELF output, this symbol points to the ELF file header (and nearby program headers) in the program's memory image.

B.2.1.1.2. Compatibility Changes

The following compatibility changes have been made since the release of **binutils** included in Red Hat Enterprise Linux 6.8:

- The **`--copy-dt-needed-entries`** command line option is no longer enabled by default. Instead, **`--no-copy-dt-needed-entries`** is now the default option.
- Evaluation of linker script expressions has been significantly improved. Note that this can negatively affect scripts that rely on undocumented behavior of the old expression evaluation.

B.2.1.2. GNU Assembler

B.2.1.2.1. New Features

The following features have been added since the release of **binutils** included in Red Hat Enterprise Linux 6.8:

- The GNU Assembler no longer requires double ampersands in macros.
- A new **`--compress-debug-sections`** command line option has been added to enable compression of DWARF debug information sections in the relocatable output file. Compressed debug sections are currently supported by the **`readelf`**, **`objdump`**, and **`gold`** tools, but not by **`ld`**.
- Support for **`.bundle_align_mode`**, **`.bundle_lock`**, and **`.bundle_unlock`** directives for x86 targets has been added..

- On x86 architectures, the GNU Assembler now allows **`rep bsf`**, **`rep bsr`**, and **`rep ret`** syntax.

B.2.1.3. Other Binary Tools

B.2.1.3.1. New Features

The following features have been added since the release of **binutils** included in Red Hat Enterprise Linux 6.8:

- The **`readelf`** and **`objdump`** tools can now display the contents of the **`.debug.macro`** sections.
- New **`--dwarf-start`** and **`--dwarf-end`** command line options have been added to the **`readelf`** and **`objdump`** tools. These options are used by the new Emacs mode (see the **`dwarf-mode.el`** file).
- A new **`--interleave-width`** command line option has been added to the **`objcopy`** tool to allow the use of the **`--interleave`** to copy a range of bytes from the input to the output.
- A new **`--dyn-syms`** command line option has been added to the **`readelf`** tool. This option can be used to dump dynamic symbol table.
- A new tool, **`elfedit`**, has been added to **binutils**. This tool can be used to directly manipulate ELF format binaries.
- A new command line option **`--addresses`** (or **`-a`** for short) has been added to the **`addr2line`** tool. This option can be used to display addresses before function and source file names.
- A new command line option **`--pretty-print`** (or **`-p`** for short) has been added to the **`addr2line`** tool. This option can be used to produce human-readable output.
- Support for **`dwz -m`** optimized debug information has been added.
- The devtoolset-2-binutils-devel package now provides the **`demangle.h`** header file.

B.2.2. Changes Since Red Hat Developer Toolset 4.0

The following features have been added since the release of **binutils** in Red Hat Developer Toolset 4.0:

- Improved security achieved through more intensive checking of the integrity of the binary files examined by the **binutils** tools. Therefore, it is much harder to make these tools crash or to attempt to read memory that does not belong to them.

B.3. CHANGES IN ELFUTILS

Red Hat Developer Toolset 4.1 is distributed with **elfutils 0.166**, which provides a number of bug fixes and feature enhancements over the version included in the previous release of Red Hat Developer Toolset. Below is a comprehensive list of new features in this release.

B.3.1. Changes Since Red Hat Developer Toolset 4.0

The following features have been added since the release of **elfutils** in Red Hat Developer Toolset 4.0:

- Support for compressed ELF sections has been added.
  - A new utility, **eu-elfcompress**, is used to compress or decompress ELF sections. The **eu-readelf** utility now has a new option, **-z** or **--decompress**, to show compressed headers and data.
  - The **libelf** library now supports various new functions, including **elf_compress**, **elf_compress_gnu**, **elf32_getchdr**, **elf64_getchdr**, and **gelf_getchdr**.
  - The **libdw** library now supports a new function, **dwelf_scn_gnu_compressed_size**.
- **pkg-config** files for the **libelf** and **libdw** libraries are now available.
- The **eu-strip** and **eu-unstrip** tools are now capable of handling ELF files with merged **strtab** and **shstrtab** tables and handling missing **SHF_INFO_LINK** section flags.
- The **dwfl_standard_find_debuginfo** function now searches subdirectories of the binary path under the debuginfo root when the separate debug file cannot be found by build-id.
- The **dwfl_linux_proc_attach** function can now be called before any **Dwfl_Modules** have been reported.

B.4. CHANGES IN GDB

Red Hat Developer Toolset 4.1 is distributed with **GDB 7.11**, which provides a number of bug fixes and improvements over the Red Hat Enterprise Linux system version and the version included in the previous release of Red Hat Developer Toolset. Below is a comprehensive list of new features in this release.

B.4.1. Changes Since Red Hat Developer Toolset 4.0

The following features have been added since the release of **GDB** in Red Hat Developer Toolset 4.0:

New Features

- Thread numbers are now per-inferior instead of global. When debugging multiple inferiors, **GDB** now displays thread IDs using a qualified **INF_NUM.THR_NUM** form. For example:

  ```
  (gdb) info threads
    Id   Target Id         Frame
    1.1  Thread 0x7ffff7fc2740 (LWP 8155) (running)
    1.2  Thread 0x7ffff7fc1700 (LWP 8168) (running)
  * 2.1  Thread 0x7ffff7fc2740 (LWP 8157) (running)
    2.2  Thread 0x7ffff7fc1700 (LWP 8190) (running)
  ```

  As consequence, thread numbers visible in the **$_thread** convenience variable and in Python's **InferiorThread.num** attribute are no longer unique between inferiors.

  GDB now maintains a second thread ID per thread, referred to as the *global thread ID*, which is the new equivalent of thread numbers in previous releases. See below for **$_gthread**.

  For backwards compatibility, thread IDs of the Machine Interpreter (MI) always refer to global IDs.

- Commands that accept thread IDs now accept the qualified **`INF_NUM.THR_NUM`** form as well. For example:

  ```
  (gdb) thread 2.1
  [Switching to thread 2.1 (Thread 0x7ffff7fc2740 (LWP 8157))]
  (running)
  (gdb)
  ```

- In commands that accept a list of thread IDs, it is now possible to refer to all threads of an inferior using a star wildcard. **GDB** accepts **`INF_NUM.*`**, to refer to all threads of inferior **`INF_NUM`**, and **`*`** to refer to all threads of the current inferior. For example, **`info threads 2.*`**.

- You can use the **`info threads -gid`** command to display the global thread ID of all threads.

- The new **`$_gthread`** convenience variable holds the global number of the current thread.

- The new **`$_inferior`** convenience variable holds the number of the current inferior.

- For multi-threaded programs, **GDB** now displays the ID and the name of the thread that hit a breakpoint or received a signal. For example:

  ```
  Thread 3 "bar" hit Breakpoint 1 at 0x40087a: file program.c, line
  20.
  Thread 1 "main" received signal SIGINT, Interrupt.
  ```

- **GDB** now allows users to specify explicit locations, bypassing the linespec parser. This feature is also available to GDB/MI clients.

- The following commands now list corresponding items in ascending ID order to maintain consistency with all other **`info`** commands: **`info threads`**, **`info inferiors`**, **`info display`**, **`info checkpoints`**, and **`maint info program-spaces`**.

- In the Ada programming language, the overloads selection menu has been enhanced to display the parameter types and the return types for the matching overloaded subprograms.

- Support has been added for thread names in the remote protocol. The reply to **`qXfer:threads:read`** may now include a name attribute for each thread.

- **GDB** now has support for fork and exec events on remote-mode Linux targets. This enables **`follow-fork-mode`**, **`detach-on-fork`**, and **`follow-exec-mode`** modes and fork and exec catchpoints.

New Commands
The following new commands have been added:

`maint set target-non-stop [on|off|auto]`, **`maint show target-non-stop`**

Controls whether **GDB** targets always operate in non-stop mode even if **`set non-stop`** is set to **`off`**. The default is **`auto`**, which means non-stop mode is enabled if it is supported by the target.

`maint set bfd-sharing`, **`maint show bfd-sharing`**

Controls whether **`bfd`** objects can be used more than once.

`set debug bfd-cache`, `show debug bfd-cache`

Controls whether debugging information regarding `bfd` caching is displayed.

`set remote multiprocess-extensions-packet`, `show remote multiprocess-extensions-packet`

Controls the use of remote protocol multiprocess extensions.

`set remote thread-events`, `show remote thread-events`

Controls the use of thread create and exit events.

`set ada print-signatures on|off`, `show ada print-signatures`

Controls the display of parameter types and return types in overloads selection menus. By default, this option is active (**on**).

`set max-value-size`, `show max-value-size`

Controls the maximum number of bytes of memory that **GDB** is allowed to allocate for value contents. The option prevents **GDB** from allocating overly large buffers by accessing inconsistent data of a debugged program. The default amount is 64 kB.

`set remote exec-event-feature-packet`, `show remote exec-event-feature-packet`

GDB now has support for exec events on extended-remote Linux targets. This enables **`follow-exec-mode`** mode and exec catchpoints. The commands control the use of the remote exec event feature.

`set remote catch-syscall-packet`, `show remote catch-syscall-packet`

GDB now has support for catching system calls on remote Linux targets. The commands control the use of the feature.

The following commands have been enhanced:

- The **`disassemble`** command now accepts a new modifier: `/s`. When the modifier is used, the command prints mixed source code and disassembly similarly to the `/m` modifier. There are two differences: disassembled instructions are now printed in program order, and source for all relevant files is now printed. The `/m` option is now considered deprecated because its "source-centric" output has not proven useful in practice.
- The **`record instruction-history`** command now accepts a new modifier: `/s`. The option behaves exactly like the `/m` modifier and prints mixed source code and disassembly.
- The **`set scheduler-locking`** command now supports a new option, **`replay`**. When **`set scheduler-locking replay`** is used, it behaves like **`off`** in record mode and like **on** in replay mode.

New Remote Packets

A number of new remote packets have been added. See Table B.1, “New Remote Packets” for a complete list.

Table B.1. New Remote Packets

Remote Packet	Description
`exec stop reason`	Serves as an indicator that an `exec` system call was executed.
`exec-events feature in qSupported`	Using the `qSupported` packet, support for exec events can be requested by **GDB** using the new `gdbfeature` exec event. The `qSupported` response can contain the corresponding `stubfeature`. Use appropriate `set` and **show** commands to enable these features and display the current setting.
`vCtrlC`	An equivalent to interrupting with the **^C** character, but it works in non-stop mode.
`thread created stop reason`	Indicates that the thread has just been created and is stopped at entry.
`thread exit stop reply`	Indicates that the thread has terminated.
`QThreadEvents`	Enables or disables thread create and exit event reporting. For example, this is used in non-stop mode when **GDB** stops a set of threads and synchronously waits for the their corresponding stop replies. Without exit events, if one of the threads exits, **GDB** would wait forever not knowing that it should no longer expect a stop for that same thread.
`N stop reply`	Indicates that there are no resumed threads left in the target (all threads are stopped). The remote stub reports support for this stop reply to the qSupported query by **GDB**.
`QCatchSyscalls`	Enables or disables catching system calls from the inferior process. The remote stub reports support for this packet to the `qSupported` query of **GDB**.
`syscall_entry stop reason`	Indicates that a system call was just called.
`syscall_return stop reason`	Indicates that a system call just returned.

Python Scripting Support

Python scripting support has been improved:

- `gdb.InferiorThread` objects now have a new attribute, `global_num`, which refers to the global thread ID. The existing **num** attribute now refers to the per-inferior number of the thread. See per-inferior thread numbers: the section called “New Features” .
- `gdb.InferiorThread` objects now have a new attribute, `inferior`, which is the `Inferior` object the thread belongs to.

Change in the Machine Interface Interpreter (GDB/MI)

The following change has been made the GDB/MI.

- The `-var-set-format` command has been enhanced to support input in zero-hexadecimal format. The output of the command is in the hexadecimal format with zero-padding on the left.

B.5. CHANGES IN SYSTEMTAP

Red Hat Developer Toolset 4.1 is distributed with **SystemTap 2.9**, which provides a number of bug fixes and feature enhancements over the version included in the previous release of Red Hat Developer Toolset. Below is a list of new features in this release.

B.5.1. Changes Since Red Hat Developer Toolset 4.0

The following features have been added since the release of **SystemTap** included in Red Hat Developer Toolset 4.0:

- The manual pages have been improved and are now more complete.
- The support for kernel backtraces without debuginfo has been improved.
- Debuginfo-related diagnostics have been improved.
- The amount of memory used by the translator has been reduced.
- Code generated by the tool now performs better.

NOTE

Incompatibility problems with old scripts can be resolved using the backward-compatibility option, `--compatible` ***`version`***, where *version* is the version of **SystemTap** for which the script was written.

B.6. CHANGES IN DYNINST

Red Hat Developer Toolset 4.1 is distributed with **dyninst 9.1.0**, which provides a number of bug fixes and feature enhancements over the version included in the previous release of Red Hat Developer Toolset. Below is a list of new features in this release.

B.6.1. Changes Since Red Hat Developer Toolset 4.0

The following features have been added since the release of **dyninst** included in Red Hat Developer Toolset 4.0:

- The new `StackMod` class now modifies the stack frame of the `BPatch_function` function.
- **Dyninst** now has a defensive mode for analyzing malware patterns.
- Gap parsing now uses machine learning to find function boundaries in a compiler-agnostic way.
- `SymtabAPI` library now provides information about inline functions.
- The `ParseAPI`, `PatchAPI`, and `BPatch` now represent loops.
- The `ProcControl` and `SymtabAPI` libraries now support thread-local storage (TLS).

- `BPatch_object` and `BPatch_module` are now used consistently for whole binaries and compilation units, respectively. Previously, a shared library was represented as `BPatch_object` and one monolithic `BPatch_module`.

- The `BPatch` library callbacks now provide `BPatch_object`.

B.7. CHANGES IN VALGRIND

Red Hat Developer Toolset 4.1 is distributed with **Valgrind 3.11.0**, which provides a number of bug fixes and enhancements over the version included in Red Hat Enterprise Linux and the previous version of Red Hat Developer Toolset.

B.7.1. Changes Since Red Hat Developer Toolset 4.0

The following bugs fixes and enhancements have been added since the release of **Valgrind** in Red Hat Developer Toolset 4.0:

- The register allocator of the **JIT** (just-in-time) compiler is now significantly faster, making the **JIT** as a whole faster. so **JIT**-intensive activities, for example, program start, are faster by approximately 5%.

- Support for the Intel AVX2 extensions is now more complete (on 64-bit targets only). On AVX2-capable hosts, the simulated CPUID now indicates AVX2 support.

- The default value for the `--smc-check` option has been changed from `stack` to `all-non-file` on targets that provide automatic D-I cache coherence (AMD64 and IBM S/390). The result is to provide, by default, transparent support for **JIT**-generated and self-modifying code on all targets.

- **Valgrind** now tries to automatically intercept user-defined alternate `malloc` and `new` allocator functions as if the program used the normal system (`glibc`) allocator. This makes it possible to use memory-tracing tools, such as **Memcheck**, on such programs out of the box. To only intercept `malloc` and `new`-related functions in system libraries, use the `--soname-synonyms=somalloc=`*`nouserintercepts`* option (where *nouserintercepts* can be any non-existing library name).

- The following enhancements have been added to the **Memcheck** component of **Valgrind**:

  - The default value for the `--leak-check-heuristics` option has been changed from `none` to `all`. This helps to reduce the number of possibly lost blocks, in particular for C++ applications.

  - The default value for the `--keep-stacktraces` has been changed from `malloc-then-free` to `malloc-and-free`. This has a small cost in memory (one word per malloc-ed block), but it allows **Memcheck** to show the three stacktraces of a dangling reference: where the block was allocated, where it was freed, and where it is acccessed after being freed.

  - The default value for `--partial-loads-ok` option has been changed from `no` to `yes`, so as to avoid false positive errors resulting from some kinds of vectorised loops.

  - A new monitor command, `xb` *`addr len`*, shows the validity bits of *len* bytes at *addr*. The `xb` monitor command is easier to use than `get_vbits` when you need to associate byte data value with their corresponding validity bits.

- The **`block_list`** monitor command has been enhanced:
  - It can now print a range of loss records.
  - It now accepts an optional argument, **`limited`** ***`max_blocks`*** to control the number of blocks printed.
  - If a block has been found using a heuristic, then **`block_list`** now shows the heuristic after the block size.
  - The loss records or blocks to be printed can be limited to the blocks found via specified heuristics.
- A new command line option, **`--expensive-definedness-checks=yes|no`**, has been added. This is useful for avoiding occasional invalid uninitialized-value errors in optimised code. Runtime degradation can be an issue because it can amount up to 25%. The slowdown is application-specific. The default setting is **no**.

APPENDIX C. REVISION HISTORY

Revision 0.0-37	**Tue 24 May 2016**	**Robert Krátký**

Release of Red Hat Developer Toolset 4.1 User Guide.

Revision 0.0-33	**Fri 13 Nov 2015**	**Robert Krátký**

Release of Red Hat Developer Toolset 4.0 User Guide.

Revision 0.0-29	**Wed 4 Nov 2015**	**Robert Krátký**

Re-release of Red Hat Developer Toolset 4.0 Beta User Guide with a section on docker-formatted container images.

Revision 0.0-28	**Wed 14 Oct 2015**	**Robert Krátký**

Release of Red Hat Developer Toolset 4.0 Beta User Guide.

Revision 0.0-25	**Thu 4 June 2015**	**Robert Krátký**

Update to reflect RHSCL 2.0 GA.

Revision 0.0-23	**Thu 23 Apr 2015**	**Robert Krátký**

Release of Red Hat Developer Toolset 3.1 User Guide.

Revision 0.0-17	**Tue 10 Mar 2015**	**Robert Krátký**

Release of Red Hat Developer Toolset 3.1 Beta User Guide.

Revision 0.0-16	**Thu 13 Nov 2014**	**Robert Krátký**

Release of Red Hat Developer Toolset 3.0 User Guide with minor post-GA fixes.

Revision 0.0-14	**Thu 30 Oct 2014**	**Robert Krátký**

Release of Red Hat Developer Toolset 3.0 User Guide.

Revision 0.0-10	**Tue 07 Oct 2014**	**Robert Krátký**

Release of Red Hat Developer Toolset 3.0 Beta-2 User Guide.

Revision 0.0-8	**Tue Sep 09 2014**	**Robert Krátký**

Release of Red Hat Developer Toolset 3.0 Beta-1 User Guide.

INDEX