

Rational® PurifyPlus

Rational® Purify®

Rational® PureCoverage®

Rational® Quantify®

Installing and Getting Started

VERSION: 2002A.06.00

PART NUMBER: 800-025784-000

UNIX

IMPORTANT NOTICE

COPYRIGHT

Copyright ©2001, 2002, Rational Software Corporation. All rights reserved.

Part Number: 800-025784-000

Version Number: 2002A.06.00

PERMITTED USAGE

THIS DOCUMENT CONTAINS PROPRIETARY INFORMATION WHICH IS THE PROPERTY OF RATIONAL SOFTWARE CORPORATION (“RATIONAL”) AND IS FURNISHED FOR THE SOLE PURPOSE OF THE OPERATION AND THE MAINTENANCE OF PRODUCTS OF RATIONAL. NO PART OF THIS PUBLICATION IS TO BE USED FOR ANY OTHER PURPOSE, AND IS NOT TO BE REPRODUCED, COPIED, ADAPTED, DISCLOSED, DISTRIBUTED, TRANSMITTED, STORED IN A RETRIEVAL SYSTEM OR TRANSLATED INTO ANY HUMAN OR COMPUTER LANGUAGE, IN ANY FORM, BY ANY MEANS, IN WHOLE OR IN PART, WITHOUT THE PRIOR EXPRESS WRITTEN CONSENT OF RATIONAL.

TRADEMARKS

Rational, Rational Software Corporation, ClearQuest, PureCoverage, Purify, Purify'd, and Quantify, among others, are either trademarks or registered trademarks of Rational Software Corporation in the United States and/or in other countries. All other names are used for identification purposes only, and are trademarks or registered trademarks of their respective companies.

PATENT

U.S. Patent Nos. 5,193,180 and 5,335,344 and 5,535,329 and 5,835,701. Additional patents pending.

Purify is licensed under Sun Microsystems, Inc., U.S. Patent No. 5,404,499.

GOVERNMENT RIGHTS LEGEND

Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in the applicable Rational Software Corporation license agreement and as provided in DFARS 277.7202-1(a) and 277.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (Oct. 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 227-14, as applicable.

WARRANTY DISCLAIMER

This document and its associated software may be used as stated in the underlying license agreement. Rational Software Corporation expressly disclaims all other warranties, express or implied, with respect to the media and software product and its documentation, including without limitation, the warranties of merchantability or fitness for a particular purpose or arising from a course of dealing, usage, or trade practice.

Contents

Preface	vii
What's in this guide?.....	vii
Audience.....	viii
Other resources.....	viii
Contacting Rational technical publications.....	viii
Contacting Rational technical support.....	viii
1 Installing the products	1
Overview.....	1
Step 1: Obtaining a license for your Rational product.....	2
Information you need to obtain a license.....	3
Obtaining a .upd import file using AccountLink.....	4
Step 2: Installing your Rational product.....	4
Information you need for rs_install.....	4
Installing the products using rs_install.....	14
Answers to questions about rs_install.....	15
Step 3: Post-installation configuration tasks.....	16
Checking and adjusting your configuration.....	20
Maintaining the rational.opt options file.....	20
Modifying the list of user names.....	21
Removing a previous product release.....	22
Requesting and installing the permanent or TLA license key.....	22
Requesting your permanent or TLA license key.....	22
Entering a permanent or TLA license key after initial installation.....	23
Supplemental notes.....	23
Creating an installation directory manually.....	23
Mounting the CD-ROM.....	24
Ejecting the CD-ROM.....	25
Using rs_install commands.....	26
Using the FLEXlm Software License Manager.....	27
2 Using Rational Purify	29
Rational Purify: What it does.....	29
Finding errors in Hello World.....	30

Instrumenting a program	31
Compiling and linking in separate stages	31
Running the instrumented program	32
Seeing all your errors at a glance	33
Finding and correcting errors	34
Understanding the cause of the error	35
Correcting the ABR error	36
Finding leaked memory	37
Correcting the MLK error	38
Looking at the heap analysis	39
Comparing program runs	40
Suppressing Purify messages	40
Saving Purify output to a view file	41
Saving a run to a view file from the Viewer	42
Opening a view file	42
Using your debugger with Purify	42
Using Purify with PureCoverage	43
Purify API functions	43
Build-time options	44
Conversion characters for filenames	45
Runtime options	45
Purify messages	47
How Purify finds memory-access errors	48
How Purify checks statically allocated memory	49
3 Using Rational PureCoverage	51
Rational PureCoverage: What it does	51
Finding untested areas of Hello World	52
Instrumenting a program	53
Running the instrumented program	54
Displaying coverage data	55
Expanding the file-level detail	56
Examining function-level detail	57
Examining the annotated source	58
Improving Hello World's test coverage	58
Using report scripts	61
Build-time options	62

Runtime options	63
Analysis-time options	63
Analysis-time mode options	64
4 Using Rational Quantify	65
Rational Quantify: What it does	65
How Quantify works: C/C++	65
How Quantify works: Java	66
Collecting performance data: C/C++	67
Interpreting the program summary: C/C++	68
Collecting performance data: Java	68
Interpreting the program summary: Java	69
Using Quantify's data analysis windows.	70
The Function List window	71
Sorting the function list.	71
Restricting functions.	72
The Call Graph window	72
Using the pop-up menu	73
Expanding and collapsing descendants.	74
The Function Detail window	74
Changing the scale and precision of data	75
Saving function detail data	75
The Annotated Source window.	76
Changing annotations	76
Saving performance data on exit	77
Comparing program runs with qxdiff.	77
Build-time options	78
qv runtime options	79
Runtime options	79
API functions: C/C++	81
API Methods: Java	82
Index	85

Preface

What's in this guide?

This guide is designed to help you get up and running quickly with Rational® PurifyPlus, Purify®, PureCoverage®, and Quantify®. It includes information about:

- Installing the products.
Note: PurifyPlus is a Rational product that includes Purify, PureCoverage, and Quantify, and provides a unified procedure for installing all three applications on your system at the same time.
- Using Purify to pinpoint runtime errors and memory leaks everywhere in your application code.
- Using PureCoverage to prevent untested application code from reaching end users.
- Using Quantify to improve the performance of your applications by finding and eliminating bottlenecks.

Purify, PureCoverage, and Quantify—the essential tools for delivering high-performance UNIX applications—use patented Object Code Insertion (OCI) technology to instrument your program, inserting instructions into the program's object code. This enables you to check your entire program, including third-party code and shared libraries, even when you don't have the source code.

Starting to use Purify, PureCoverage, and Quantify is as easy as adding the product name (`purify`, `purecov`, or `quantify`) to the front of your link command line. For example:

```
% purify cc -g hello_world.c
```

Audience

Read this guide if you are responsible for installing Rational PurifyPlus, Purify, PureCoverage, or Quantify, or if you need an introduction to the use of Purify, PureCoverage, or Quantify.

Other resources

- A complete online help system is available for each application. Select **Help > Help topics**.

For help with a window, select **Help > On window**. For help with a specific menu item or control button in a window, select **Help > On context**, then click the menu item or control button.

Note: You can also view the help systems independently of the products. Open the following in your Netscape browser:

- `'purify -printhomedir' /UI/html/punix.htm`
 - `'purecov -printhomedir' /UI/html/pcu.htm`
 - `'quantify -printhomedir' /UI/html/qunix.htm`
- For information about Rational Software and Rational Software products, go to <http://www.rational.com>.

Contacting Rational technical publications

Please send any feedback about this documentation to the Rational technical publications department at techpubs@rational.com.

Contacting Rational technical support

You can contact Rational technical support by e-mail at support@rational.com.

You can also reach Rational technical support over the Web or by telephone. For contact information, as well as for answers to common questions about Purify, PureCoverage, and Quantify, go to <http://www.rational.com/support>.

Overview

The Rational® PurifyPlus product family includes Rational PurifyPlus, Rational Purify®, Rational PureCoverage®, and Rational Quantify®.

A PurifyPlus license allows you to use Purify, PureCoverage, and Quantify, and provides a unified procedure for installing all three applications on your system at the same time. Individual product licenses are also available.

Installing each product includes three steps, though depending on your situation you may be able to skip one or more of them:

- 1 Obtaining a license for the product.
- 2 Installing the product on your system using the `rs_install` program.
- 3 Performing any necessary post-installation configuration procedures.

This chapter tells you how to gather the information you need to perform these steps, as well as basic instructions to get you started with each one. It also contains information about related tasks (such as uninstalling) and administering the GLOBEtrotter FLEXIm® Software License Manager that is included with your Rational Software product.

Step 1: Obtaining a license for your Rational product

Refer to the following table to determine whether you need to perform Step 1, based on the type of license you are setting up.

License Type	Description	Instructions
Permanent	Allows the use of the product without time limits.	<p>If you do not yet have a valid permanent or TLA license set up, gather the information specified in <i>Information you need to obtain a license</i> on page 3 and then request a license from AccountLink, Rational Software's license management web tool. This applies if you are a first-time purchaser of the Rational PurifyPlus products.</p> <p>If you are upgrading from an earlier version of the product and still have a valid permanent or TLA license, go right to <i>Step 2: Installing your Rational product</i> on page 4. When you run the installation script, be sure to choose the option USE AN EXISTING RATIONAL LICENSE; you must then enter the location of your existing license.</p> <p>To enter a permanent or TLA license key after you've been using a startup or evaluation license, see <i>Entering a permanent or TLA license key after initial installation</i> on page 23.</p>
Term License Agreement (TLA)	Allows the use of the product for a specific period of time.	
Startup	Gets you up and running as soon as you receive your Rational product. Note that you will have to enter a permanent or TLA key later to continue using the product.	<p>Go right to <i>Step 2: Installing your Rational product</i> on page 4.</p> <p>When you receive your permanent or TLA license, see <i>Entering a permanent or TLA license key after initial installation</i> on page 23.</p>
Evaluation	Allows you to try out the product for a limited time. Note that you will have to enter a permanent or TLA key later to continue using the product.	

Information you need to obtain a license

In addition to contact information, you will need to provide the following to AccountLink, Rational Software's license management web tool, in order to obtain a .upd import file with licensing information:

Data for AccountLink	Notes	Your Entry
Your Rational account number	Source: your Rational license key certificate.	
License Type	Source: your Rational license key certificate Permanent and TLA licenses for PurifyPlus can be either Floating licenses or Named User licenses. Permanent and TLA licenses for Purify, PureCoverage, and Quantify are always Named User licenses.	
Rational Product Line	Source: your Rational license key certificate	DEVELOPER TOOLS
Product Name	Source: your Rational license key certificate.	PurifyPlus for UNIX, Purify for UNIX, PureCoverage for UNIX, or Quantify for UNIX
Quantity	Source: your Rational license key certificate. This is the number of Floating licenses or Named User licenses that you intend to install. This cannot be greater than the number of licenses you purchased.	
Host Name and hostid	This is the name and hostid of the machine that you intend to use as your license server host. If the license server host is different from the installation machine, you must have remote shell access from the installation machine to the license server host during installation. In addition, the installation directory must be accessible from the license server host. If you do not know the hostid of your license server host, you can download and run the tool <code>get_hostinfo.sh</code> after you have begun your license request on AccountLink; go to <i>Obtaining a .upd import file using AccountLink</i> on page 4.	

Obtaining a .upd import file using AccountLink

Access AccountLink at <http://www.rational.com/accountlink>. The AccountLink website provides instructions for requesting licenses.

When you supply the required information to AccountLink, you will receive by return email a file named

`license_for_<server name>.upd`. Save this file as a text file in a location that is accessible from the installation machine.

Go on to *Step 2: Installing your Rational product* on page 4.

Step 2: Installing your Rational product

Install your product using the script `rs_install`.

Information you need for `rs_install`

The information you need to supply to the `rs_install` script depends on what type of license you have, and on how you are setting it up:

- If you are installing a new permanent or TLA license, and you have a `.upd` file that you can use to import licensing data, go to page 5.
- If you are installing a new permanent or TLA license, and you **do not** have a `.upd` file that you can use to import licensing data, go to page 8.
- If you already have a permanent or TLA license set up, go to page 10.
- If you are installing a temporary or evaluation license, go page 12.

If you are installing a new permanent or TLA license and are importing a .upd file

Gather the information specified in the following table:

Data for rs_install	Notes	Your Entry
<p>The full pathname to the installation location (referred to in this chapter as Rational).</p>	<p>This is the directory for installing all Rational Software products. You must have 20 megabytes of free disk space for each installation of Purify, Quantify, and PureCoverage. You must have 60 megabytes for a complete installation of PurifyPlus. Note that these are per-platform figures.</p> <p>The directory must be accessible from every machine on which you plan to run the Rational products—both the machines on which users <i>instrument</i> their applications, and the machines on which users <i>run</i> their applications. The directory must be the same for each machine, so you cannot use a local automount path like <code>/tmp_mnt/Rational</code>.</p> <p>If <code>Rational</code> does not already exist, the installation program will create it when you enter the full pathname.</p> <p>If you are installing on a read-only file system, or want to create this directory manually, see <i>Supplemental notes</i> on page 23. This section also shows the structure of the directory after installation.</p>	

Data for rs_install	Notes	Your Entry
<p>Note: If you are installing a Named User license, you must supply user names for each individual who will be using the product. You must include your user name in order to perform the post-installation self-test successfully. User names are recorded in the FLEXlm options file, <code>rational.opt</code>. For information about the options file, see <i>Maintaining the rational.opt options file</i> on page 20.</p> <p>You do not need this information if you are installing a Floating license for PurifyPlus.</p> <p>To input User names, you need the data in A or B below, or the data in A supplemented with the data in B.</p>		
<p>A. Path of the PureLA directory containing the file <code>users.purela</code> (available only if you licensed an earlier version of the product using PureLA License Advisor).</p>	<p>If you are currently running the product under a PureLA license, you have the option of importing the user names from the PureLA database instead of entering them manually. The PureLA directory is located in the same parent directory as the previous product installation, which you can find with the command <code><product> -printhomedir</code>, where <code><product></code> is <code>purify</code>, <code>purecov</code>, or <code>quantify</code>.</p> <p>You can modify the list of imported user names, either while you're running <code>rs_install</code> or afterwards. If the number of user names is not the same as the number of licenses you bought, <code>rs_install</code> will help you correct the list.</p>	
<p>B. user names (all names; or some or none, in combination with an option to generate dummy names).</p>	<p>You can enter all user names. The number of names you enter must match the number of licenses you purchased.</p> <p>You can enter some user names, and then enter <code>-n</code> to populate the rest of the options file with dummy names as placeholders that you can replace later.</p> <p>Or you can just enter <code>-n</code> to enter nothing but dummy names, and update the options file later.</p>	
<p>Name for the Rational license file (<code><server name>.dat</code>), including full pathname.</p>	<p>The <code>rs_install</code> program, which creates this file when it runs, will suggest <code><server name>.dat</code> as a default.</p> <p>See <i>The Rational license file</i> on page 27 for information about the file.</p>	
<p>Note: If you have installed Rational ClearQuest, Rational Software's change request management system, it is possible for Purify and PureCoverage users to file change requests from within the Purify and PureCoverage graphical interface. To configure this integration between the products:</p> <ul style="list-style-type: none"> ▪ You as the installer can provide <code>rs_install</code> with site-wide default values. Note that individual users can override the default values if they have different requirements, following the instructions in the Purify and PureCoverage online help systems. (See "ClearQuest" in the online help index.) ▪ Alternatively, you can choose not to enter default values. In this case, users must enter their own values in order to use the integration. <p>The source for the following configuration values is your ClearQuest system administrator.</p>		

Data for rs_install	Notes	Your Entry
ClearQuest client interface.	<p>If you are installing on a Solaris or HP-UX platform, you can specify the default ClearQuest client for your users:</p> <ul style="list-style-type: none"> ▪ x for the ClearQuest native X client ▪ web for the ClearQuest web client <p>All other platforms support only the web client.</p> <p>Users must have ClearQuest on their path in order to use the native X client.</p>	
Name of the default ClearQuest database.	<p>Enter the name of a site-wide default ClearQuest database for your users who access ClearQuest through the native X client.</p> <p>Even if you have chosen web as your default interface, you may want to enter a value here for users who prefer to use the ClearQuest native X client.</p> <p>Source: ClearQuest system administrator.</p>	
URL for the ClearQuest web client.	<p>Specify the URL if you selected web as your default ClearQuest client interface.</p> <p>Even if you have chosen x as your default interface, you may want to enter a value here for users who prefer to use the web interface.</p> <p>Source: ClearQuest system administrator.</p>	

When you have gathered the information you need, go on to *Installing the products using rs_install* on page 14.

If you are installing a new permanent or TLA license without importing a .upd file

Gather the information specified in the following table:

Data for rs_install	Notes	Your Entry
The full pathname to the installation location (referred to in this chapter as Rational).	<p>This is the directory for installing all Rational Software products. You must have 20 megabytes of free disk space for each installation of Purify, Quantify, and PureCoverage. You must have 60 megabytes for a complete installation of PurifyPlus. Note that these are per-platform figures.</p> <p>The directory must be accessible from every machine on which you plan to run the Rational products—both the machines on which users <i>instrument</i> their applications, and the machines on which users <i>run</i> their applications. It must be the same for each machine, so you cannot use a local automount path like <code>/tmp_mnt/Rational</code>.</p> <p>If Rational does not already exist, the installation program will create it when you enter the full pathname.</p> <p>If you are installing on a read-only file system, or if you want to create this directory manually, see <i>Supplemental notes</i> on page 23. This section also shows you the structure of the directory after installation.</p>	
Host name or IP address of the host machine on which the license server is to run (the "license server host").	<p>If the license server host is different from the installation machine, you should have remote shell access from the installation machine to the license server host. If you do you have remote shell access, the <code>rs_install</code> program provides instructions for how to proceed.</p> <p>In addition, the installation directory must be accessible from the license server host.</p>	
License server port number.	<p>This is the port at which the license server listens for license requests. Default is 27000. You can use any port number that is not already in use. The <code>/etc/services</code> file on the license host lists all ports in use by most commonly used services, but other ports may be in use on your system as well. FLEXlm reserves ports 27000–27009 for its use; these ports are ordinarily available unless a different FLEXlm server on the license host is using them.</p> <p>The <code>rs_install</code> program checks to make sure that the license server port number does not conflict with entries in the <code>/etc/services</code> file on the license server host, or with NIS services.</p>	
License quantity.	Source: your Rational license key certificate.	

Data for rs_install	Notes	Your Entry
<p>Note: If you are installing a Named User license, you must supply user names for each individual who will be using the product. You must include your user name in order to perform the post-installation self-test successfully. User names are recorded in the FLEXlm options file, <code>rational.opt</code>. For information about the options file, see <i>Maintaining the rational.opt options file</i> on page 20.</p> <p>You do not need this information if you are installing a Floating license for PurifyPlus.</p> <p>To input User names, you need the data in A or B below, or the data in A supplemented with the data in B.</p>		
<p>A. Path of the PureLA directory containing the file <code>users.purela</code> (available only if you licensed an earlier version of the product using PureLA License Advisor).</p>	<p>If you are currently running the product under a PureLA license, you have the option of importing the user names from the PureLA database instead of entering them manually. The PureLA directory is located in the same parent directory as the previous product installation, which you can find with the command <code><product> -printhomedir</code>, where <code><product></code> is <code>purify</code>, <code>purecov</code>, or <code>quantify</code>.</p> <p>You can modify the list of imported user names, either while you're running <code>rs_install</code> or afterwards. If the number of user names is not the same as the number of licenses you bought, <code>rs_install</code> will help you correct the list.</p>	
<p>B. user names (all names; or some or none, in combination with an option to generate dummy names).</p>	<p>You can enter all user names. The number of names you enter must match the number of licenses you purchased.</p> <p>You can enter some user names, and then enter <code>-n</code> to populate the rest of the options file with dummy names as placeholders that you can replace later.</p> <p>Or you can just enter <code>-n</code> to enter nothing but dummy names, and update the options file later.</p>	
<p>Name for the Rational license file (<code><server name>.dat</code>), including full pathname.</p>	<p>The <code>rs_install</code> program, which creates this file when it runs, will suggest <code><server name>.dat</code> as a default.</p> <p>If you want to use an existing Rational license <code>.dat</code> file, enter its name, including full pathname, instead of the default. The <code>rs_install</code> program makes a backup of the existing license file before it processes the file with the new data. For information, see <i>The Rational license file</i> on page 27.</p>	
<p>Note: If you have installed Rational ClearQuest, Rational Software's change request management system, it is possible for Purify and PureCoverage users to file change requests from within the Purify and PureCoverage graphical interface. To configure this integration between the products:</p> <ul style="list-style-type: none"> ▪ You as the installer can provide <code>rs_install</code> with site-wide default values. Note that individual users can override the default values if they have different requirements, following the instructions in the Purify and PureCoverage online help systems. (See "ClearQuest" in the online help index.) ▪ Alternatively, you can choose not to enter default values. In this case, users must enter their own values in order to use the integration. <p>The source for the following configuration values is your ClearQuest system administrator.</p>		

Data for rs_install	Notes	Your Entry
ClearQuest client interface.	<p>If you are installing on a Solaris or HP-UX platform, you can specify the default ClearQuest client for your users:</p> <ul style="list-style-type: none"> ▪ x for the ClearQuest native X client ▪ web for the ClearQuest web client <p>All other platforms support only the web client.</p> <p>Users must have ClearQuest on their path in order to use the native X client.</p>	
Name of the default ClearQuest database.	<p>Enter the name of a site-wide default ClearQuest database for your users who access ClearQuest through the native X client.</p> <p>Even if you have chosen web as your default interface, you may want to enter a value here for users who prefer to use the ClearQuest native X client.</p> <p>Source: ClearQuest system administrator.</p>	
URL for the ClearQuest web client.	<p>Specify the URL if you selected web as your default ClearQuest client interface.</p> <p>Even if you have chosen x as your default interface, you may want to enter a value here for users who prefer to use the web interface.</p> <p>Source: ClearQuest system administrator.</p>	

When you have gathered the information you need, go on to *Installing the products using rs_install* on page 14.

If you already have a permanent or TLA license set up

If you are upgrading from an earlier version of the product and still have a valid permanent license, most of the information you need is already available in your system. When you run the `rs_install` program, be sure to choose the option **USE AN EXISTING RATIONAL LICENSE**.

Gather the information specified in the following table:

Data for rs_install	Notes	Your Entry
Full pathname of your license file, <server name>.dat.	Alternatively, you can supply the host name or IP address of the host machine on which the license server is running (the "license server host") and the license server port number.	

Data for <code>rs_install</code>	Notes	Your Entry
<p>The full pathname to the current installation location, or to the new installation location (referred to in this chapter as Rational).</p>	<p>This is the directory for installing all Rational Software products. You can use your existing installation; but if you use a new directory, it must meet the following requirements:</p> <ul style="list-style-type: none"> ▪ You must have 20 megabytes of free disk space for each installation of Purify, Quantify, and PureCoverage. You must have 60 megabytes for a complete installation of PurifyPlus. Note that these are per-platform figures. ▪ The directory must be accessible from every machine on which you plan to run the Rational products—both the machines on which users <i>instrument</i> their applications, and the machines on which users <i>run</i> their applications. It must be the same for each machine, so you cannot use a local automount path like <code>/tmp_mnt/Rational</code>. <p>If the directory you specify does not already exist, the installation program will create it when you enter the full pathname.</p> <p>If you are installing on a read-only file system, or if you want to create this directory manually, see <i>Supplemental notes</i> on page 23. This section also shows you the structure of the directory after installation.</p>	
<p>Note: If you have installed Rational ClearQuest, Rational Software's change request management system, it is possible for Purify and PureCoverage users to file change requests from within the Purify and PureCoverage graphical interface. To configure this integration between the products:</p> <ul style="list-style-type: none"> ▪ You as the installer can provide <code>rs_install</code> with site-wide default values. Note that individual users can override the default values if they have different requirements, following the instructions in the Purify and PureCoverage online help systems. (See "ClearQuest" in the online help index.) ▪ Alternatively, you can choose not to enter default values. In this case, users must enter their own values in order to use the integration. <p>The source for the following configuration values is your ClearQuest system administrator.</p>		
<p>ClearQuest client interface.</p>	<p>If you are installing on a Solaris or HP-UX platform, you can specify the default ClearQuest client for your users:</p> <ul style="list-style-type: none"> ▪ <code>x</code> for the ClearQuest native X client ▪ <code>web</code> for the ClearQuest web client <p>All other platforms support only the web client.</p> <p>Users must have ClearQuest on their path in order to use the native X client.</p>	
<p>Name of the default ClearQuest database.</p>	<p>Enter the name of a site-wide default ClearQuest database for your users who access ClearQuest through the native X client.</p> <p>Even if you have chosen <code>web</code> as your default interface, you may want to enter a value here for users who prefer to use the ClearQuest native X client.</p> <p>Source: ClearQuest system administrator.</p>	

Data for rs_install	Notes	Your Entry
URL for the ClearQuest web client.	Specify the URL if you selected web as your default ClearQuest client interface. Even if you have chosen x as your default interface, you may want to enter a value here for users who prefer to use the web interface. Source: ClearQuest system administrator.	

When you have gathered the information you need, go on to *Installing the products using rs_install* on page 14.

If you are installing a temporary or evaluation license

Gather the information specified in the following table:

Data for rs_install	Notes	Your Entry
The full pathname to the installation location (referred to in this chapter as Rational).	This is the directory for installing all Rational Software products. You must have 20 megabytes of free disk space for each installation of Purify, Quantify, and PureCoverage. You must have 60 megabytes for a complete installation of PurifyPlus. Note that these are per-platform figures. The directory must be accessible from every machine on which you plan to run the Rational products—both the machines on which users <i>instrument</i> their applications, and the machines on which users <i>run</i> their applications. It must be the same for each machine, so you cannot use a local automount path like /tmp_mnt/Rational. If Rational does not already exist, the installation program will create it when you enter the full pathname. If you are installing on a read-only file system, or if you want to create this directory manually, see <i>Supplemental notes</i> on page 23. This section also shows you the structure of the directory after installation.	
License key type.	Source: your Rational license key certificate or email from Rational Software.	startup or evaluation
Expiration date.	Source: your Rational license key certificate or email from Rational Software. If you have a startup or evaluation license, enter the date in the dd-mmm-yyyy format. The field is not case sensitive.	

Data for <code>rs_install</code>	Notes	Your Entry
<p>Note: If you have installed Rational ClearQuest, Rational Software's change request management system, it is possible for Purify and PureCoverage users to file change requests from within the Purify and PureCoverage graphical interface. To configure this integration between the products:</p> <ul style="list-style-type: none"> ▪ You as the installer can provide <code>rs_install</code> with site-wide default values. Note that individual users can override the default values if they have different requirements, following the instructions in the Purify and PureCoverage online help systems. (See "ClearQuest" in the online help index.) ▪ Alternatively, you can choose not to enter default values. In this case, users must enter their own values in order to use the integration. <p>The source for the following configuration values is your ClearQuest system administrator.</p>		
ClearQuest client interface.	<p>If you are installing on a Solaris or HP-UX platform, you can specify the default ClearQuest client for your users:</p> <ul style="list-style-type: none"> ▪ <code>x</code> for the ClearQuest native X client ▪ <code>web</code> for the ClearQuest web client <p>All other platforms support only the web client. Users must have ClearQuest on their path in order to use the native X client.</p>	
Name of the default ClearQuest database.	<p>Enter the name of a site-wide default ClearQuest database for your users who access ClearQuest through the native X client. Even if you have chosen <code>web</code> as your default interface, you may want to enter a value here for users who prefer to use the ClearQuest native X client. Source: ClearQuest system administrator.</p>	
URL for the ClearQuest web client.	<p>Specify the URL if you selected <code>web</code> as your default ClearQuest client interface. Even if you have chosen <code>x</code> as your default interface, you may want to enter a value here for users who prefer to use the web interface. Source: ClearQuest system administrator.</p>	

When you have gathered the information you need, go on to *Installing the products using `rs_install`* on page 14.

Installing the products using `rs_install`

For information about specific product and operating system versions, see the `README` file on your CD-ROM or in the directory that results when you unpack the `tar` file from Rational Software.

To install the products:

1 Make the product available for installation.

If you are installing the product from the Rational Software product CD-ROM and need instructions for mounting a CD-ROM, see page 24.

If you have obtained the products from a web or FTP download, unpack the compressed `tar` file. The directory that is created is equivalent to the top level of the CD-ROM.

2 Run the `rs_install` program. The `rs_install` program is a complete installer that guides you through the following processes:

- Setting up the license server.
- Installing product licenses.
- Installing the selected product and documentation.

Note: Your users can get online help only if you install the `html` documentation.

- Performing the post-installation tasks.

To run the `rs_install` program, go to the directory where you mounted the CD-ROM or unpacked your `tar` file. (You should not be `root` when you run `rs_install`.) For example:

```
# exit
% cd /cdrom
% ./rs_install
```

The `rs_install` program prompts you through the installation, providing detailed instructions along with default settings. The defaults appear in brackets, for example `[2]`. To accept the default, press `ENTER`.

Note: After you install your license key, the `rs_install` program reminds you that you must configure your server to automatically restart the license server when it reboots. The `rs_install` program gives you instructions for doing this.

- 3 When installation is complete, go to *Step 3: Post-installation configuration tasks* on page 16 and perform any necessary post-installation procedures.

Answers to questions about `rs_install`

Below are the answers to some common questions about the `rs_install` program.

- **Can I rerun parts of the installation?** Yes. The `rs_install` program provides commands that enable you to rerun specific sections of the installation as needed. See *Using rs_install commands* on page 26.
- **Do I have to reenter my license server information each time I install a product?** No. You only need to enter this information once. The `rs_install` program saves the information you enter about yourself and about the machine to be used as the license server for your Rational Software product licenses in two text files: an `rs_install.defaults` file that contains information about you and your license server, and a file such as `rs_install.PurifyPlus.2002a.06.00` that records product-specific information. The `rs_install` program reports the location of these files when you quit the program. The next time you run `rs_install`, the program uses the saved configuration information.
To change your license server, use the `license_setup` command; see *Using rs_install commands* on page 26.
- **Do I need to install all my licenses on one server?** No. You are not required to use all of your allowed licenses for a single license server. You might want to install a product at another site and configure a license server at that site to serve the remaining licenses in your Rational Software account.
- **Which type of product license key should I install?** If you already have your permanent or TLA license key, you can install it right away. You can also request a permanent license key at www.rational.com/accountlink. Otherwise, select the startup or evaluation license to get started using the product.
Note: To ensure uninterrupted use of your Rational Software product, you should install your permanent or TLA license key as soon as possible.

- **Can I import existing user names from an earlier installation of the product installed under PureLA License Administrator with Named User licensing?** Yes. The `rs_install` script asks you if you want to import the existing `users.purela` file, and also permits you to edit the imported user names. You can also edit the user names after installation; see *Maintaining the rational.opt options file* on page 20.
- **The host I want to use as the license server for my new products is already the license server for other Rational products. How do I share the server?** You must add the new licenses to your current Rational `.dat` license file. To do this, specify the current Rational `.dat` license file as the license file name instead of using the default.
- **How do I get updates for the `rs_install` program and for the Rational products?** You can get updates from within the `rs_install` program, though you must be running the program on a machine that has internet access. The `rs_install` program's Licensing Options screen lets you select an item to download the latest version of `rs_install` (in which case `rs_install` replaces itself and restarts using the new version) or get product updates.
- **How do I report problems or make suggestions for the `rs_install` program?** You can submit comments by running the `rs_install` command with the `-report` option. This option helps you organize installation and licensing information and e-mail it to Rational technical support. You can, of course, also call or e-mail Rational technical support without this option.

Step 3: Post-installation configuration tasks

Configuration tasks include tasks that the `rs_install` program performs (or helps you perform) and tasks that you must complete manually.

The `rs_install` program performs its configuration tasks by calling the `<product>.configure` command for the product you are installing; `<product>` is `purify`, `purecov`, or `quantify`. You can also rerun the `<product>.configure` command at any time to check or to adjust your configuration; refer to *Checking and adjusting your configuration* on page 20.

The tasks that `<product>.configure` performs include:

- Configuring the cache directory.
- Setting up the online help system.
- Integrating Rational ClearQuest (for PurifyPlus, Purify, and PureCoverage).
- Running `<product>_test` to validate setup.

The tasks that you must perform manually include:

- Installing on a read-only file system; see page 17.
- Making the manual pages available; see page 18.
- Making the products available to all users; see page 18.

Note that `<producthome>` is the home directory of Purify, PureCoverage, or Quantify.

Installing on a read-only file system

Purify, PureCoverage, and Quantify work by creating and monitoring special instrumented versions of object files and libraries. They must be able to write these instrumented files to a cache directory, which by default is `Rational/releases/<producthome>/cache`.

For this reason, if you install any of the products on a file system that is mounted read-only by client machines, you must create symbolic links to a writable file system. The `rs_install` program guides you through the process of selecting a shared directory that is mounted read/write on client machines and linking the `cache` directory to this publicly writable directory.

If there is no writable shared directory mounted on client machines, have all users make a `cache` subdirectory in their home directory and set the product's `-cache-dir` option to this directory. For example:

```
% mkdir $HOME/cache
% echo $PUREOPTIONS
```

If the `PUREOPTIONS` environment variable is already set, have users specify the `-cache-dir` option:

```
csh % setenv PUREOPTIONS "-cache-dir=$HOME/cache \
    $PUREOPTIONS"
sh, ksh $ PUREOPTIONS="-cache-dir=$HOME/cache \
    $PUREOPTIONS"; export PUREOPTIONS
```

If the `PUREOPTIONS` environment variable is *not* set, have users specify:

```
csch % setenv PUREOPTIONS "-cache-dir=$HOME/cache"
sh, ksh $ PUREOPTIONS="-cache-dir=$HOME/cache"; export \
    PUREOPTIONS
```

Have all users add this same specification to their local or central `.cshrc` file, or its equivalent.

Making the manual pages available

The `rs_install` program installs the product manual pages in `Rational/releases/<producthome>/man`, where `<producthome>` is the home directory for Purify, PureCoverage, or Quantify. To make the manual pages available, do one of the following:

- Set your `MANPATH` environment variable to include `Rational/releases/<producthome>/man`.
- Copy the manual pages for the product into your `man` directory. If necessary, log in as `root` to do this.

Making the products available to all users

Note: If you are using Named User licensing, users must be listed in the `rational.opt` file in order to use Purify, PureCoverage, and Quantify; to add users to the options file, see *Maintaining the rational.opt options file* on page 20.

To make the products available to all users listed in `rational.opt`, add the full `Rational/releases/<producthome>` pathname to each user's `PATH` environment variable, or specify the full pathname in makefiles.

As an alternative to modifying your `PATH` environment variable, you can create a symbolic link to `<producthome>/<product>` from a directory such as `/usr/local/bin`. Make sure this is a symbolic link, not a copy or a hard link. Create symbolic links for each product you install, as in the following examples:

- For Purify:

```
% rm /usr/local/bin/purify
% ln -s Rational/releases/\
    <producthome>/purify /usr/local/bin
```

- For PureCoverage:

```
% rm /usr/local/bin/purecov
% ln -s Rational/releases/\
    <producthome>/purecov /usr/local/bin
```

For PureCoverage, you also need to create symbolic links to the `pc_*` script files:

```
% rm -i /usr/local/bin/pc_*
% ln -s Rational/releases/\
    <purecovhome>/scripts/pc_* /usr/local/bin
```

For more information on the `pc_*` scripts, see the PureCoverage online help system.

- For Quantify:

```
% rm /usr/local/bin/quantify
% ln -s Rational/releases/\
    <producthome>/quantify /usr/local/bin
```

For Quantify, you also need to create symbolic links to the `qv` program and to the `qx` script files:

```
% rm /usr/local/bin/qv
% rm -i /usr/local/bin/qx*
% ln -s Rational/releases/\
    <quantifyhome>/qv /usr/local/bin
% ln -s Rational/releases/\
    <quantifyhome>/qx* /usr/local/bin
```

For more information on the `qv` program and on the `qx` scripts, see the Quantify online help system.



- Create symbolic links for debugger scripts on HP-UX:

On HP-UX, Purify, PureCoverage, and Quantify include three scripts that enable you to start instrumented programs under a debugger. You need to create symbolic links to these scripts. For example, for Purify:

```
% rm /usr/local/bin/purify_dde
% rm /usr/local/bin/purify_xdb
% rm /usr/local/bin/purify_softdebug
% ln -s <purifyhome>/purify_dde /usr/local/bin
% ln -s <purifyhome>/purify_xdb /usr/local/bin
% ln -s <purifyhome>/purify_softdebug /usr/local/bin
```

For PureCoverage and Quantify, create the same symbolic links, substituting `purecov` or `quantify` for `purify`.

The installation is now complete. To add names to the options file, see *Maintaining the rational.opt options file* on page 20. To remove previous versions of the products, see *Removing a previous product release* on page 22.

Checking and adjusting your configuration

You can run the script `<product>.configure` (where `<product>` is `purify`, `purecov`, or `quantify`) at any time to check that your configuration is correct and to make adjustments.

To use the script, go to the product home directory. For example, for Purify, type:

```
% cd `purify -printhomedir`
```

To check your configuration and licensing, type:

```
% ./purify.configure -check
```

To run the script in interactive mode, type:

```
% ./purify.configure
```

You can also run the script in batch mode, specifying the parameters you want changed as arguments to options. For a list and description of batch mode options, type:

```
% ./purify.configure -help
```

Maintaining the rational.opt options file

Named User licensing is always used with Purify, PureCoverage, and Quantify, and is available for use with PurifyPlus. Under *Named User* licensing, the user names of all users who are authorized to run Purify, PureCoverage, and Quantify must be listed in the `rational.opt` options file. The number of user names in the file must match the number of licenses you have installed.

Users who are identified in the file can use all features of the product, including instrumenting applications, running instrumented applications, and viewing saved data files in the product's user interface. A user can run as many concurrent sessions as desired on a single host machine; this consumes a single license. The same user can run the product on additional host machines, but consumes another license for each additional machine.

The options file is created when you run the `rs_install` program. By default, this file is `Rational/config/rational.opt`. You can relocate the file yourself after installation, provided that you edit the license file `DAEMON` line to specify the new path:

```
DAEMON rational /etc/rational /mydir/rational.opt
```

During installation, `rs_install` asks you to supply user names, one for each license you purchased. You don't have to enter all user names during installation; `rs_install` will generate dummy names to bring the total up to the number of licenses you purchased. Your entries—real names, automatically generated dummy names, or both—are recorded in the options file.

The user names are recorded in the options file in `GROUP` directives. An `INCLUDE` directive follows each `GROUP` directive, specifying one product that the users in the group are authorized to use:

```
GROUP <group name> <user1> <user2> . . . <usern>
INCLUDE <product>:KEY=<license key> GROUP <group name>
```

For example, in the following, `alice`, `tom`, and `harry` can use `Purify`, but only `alice` and `harry` can use `Quantify`:

```
GROUP DevTools1 alice tom harry
INCLUDE purify:KEY=456778982 GROUP DevTools1
GROUP DevTools2 alice harry
INCLUDE quantify:KEY=12345778654 GROUP DevTools2
```

The `KEY` is the license key from your `.dat` license file.

Modifying the list of user names

Note: If you modify the options file while the license vendor daemon is running, you must restart the license server.

You can add, change, or delete user names by running the `options_setup` script. You can also add, change, or delete user names in the options file using any text editor.

The number of users listed for each product must always match the number of licenses that you purchased. The license server must be restarted before the changes can take effect; the `options_setup` script restarts the license server for you.

For additional information about the options file, refer to your `FLEXlm` user's manual.

Removing a previous product release

Note: Only the installer of the product can uninstall it.

After you install the latest version of Purify, PureCoverage, or Quantify, and after all users have switched to the new version, you can remove the old release to reclaim disk space.

To remove a previous release of Purify, PureCoverage, or Quantify, go to the `Rational` directory and run the `uninstall` script:

```
% cd Rational
% config/uninstall
```

Running the `uninstall` script with no command-line arguments causes it to display the list of products in the `releases` directory. The script prompts you for the product you want to remove.

Requesting and installing the permanent or TLA license key

When you purchase Purify, PureCoverage, or Quantify, you purchase a specific number of licenses for each product. Rational Software issues you a license key for the product that corresponds to the type and number of licenses you purchased. You need this license key to use the software.

Purify, PureCoverage, and Quantify come with a startup license that you can use to get started using the product. You then request a permanent or TLA license key from Rational Software at www.rational.com/accountlink and install it to ensure continued use of the product. The startup license key and other licensing information is available from the License Key Certificate included in the product packaging.

Purify, PureCoverage, and Quantify use the FLEXlm Software License Manager from GLOBETrotter Software, Inc. to manage product licenses. For more information on FLEXlm, see *Using the FLEXlm Software License Manager* on page 27.

Requesting your permanent or TLA license key

To request a permanent or TLA license key, go to www.rational.com/accountlink and follow the instructions provided there.

Entering a permanent or TLA license key after initial installation

To enter your permanent or TLA license key after you have installed your Rational Software product and exited the `rs_install` program:

- 1 Go to the `Rational/releases/PurifyPlusFamily.<version>` directory and run the `license_setup` program. For instructions, see *Using rs_install commands* on page 26.
- 2 For the licensing option, select the option for setting up a permanent license.

Note: The program tells you how to update your license server machine so that it restarts the license server when it reboots. You need root permission to perform the update.

Supplemental notes

Creating an installation directory manually

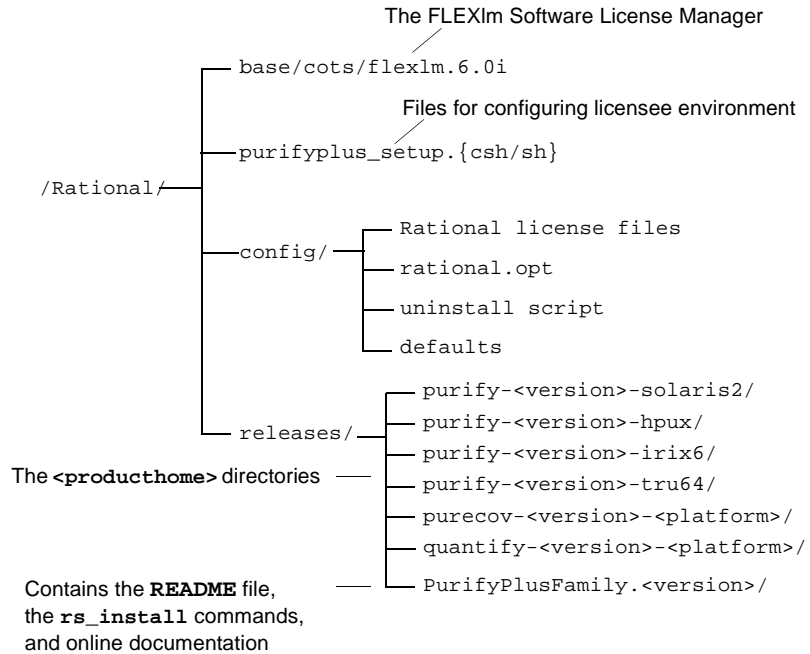
You need a publicly readable directory for the installation of Purify, PureCoverage, and Quantify. If one does not already exist, you can create it when you run `rs_install`. You can also create it manually before you start `rs_install`.

- 1 Log into a UNIX workstation that provides access to the product files to be installed, and that mounts the file system(s) into which you want to load the product.
- 2 Create a `Rational` directory. For example:

```
% mkdir /opt/Rational
```

The `Rational` directory must be visible on all machines that are to run this product. The NFS name for `Rational` must be the same on all machines. (If you are installing the product for your use only, you can install it in your home directory.)

After the installation, the Rational directory is structured like this:



Note: Purify, PureCoverage, and Quantify must be able to write instrumented files to a `cache` subdirectory of the `<producthome>` directory. If you install on a read-only file system, you must create symbolic links to a writable file system. See *Installing on a read-only file system* on page 17.

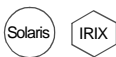
Mounting the CD-ROM

The following instructions refer to specific operating systems.

To determine your operating system, type:

```
% uname -a
```

Before you begin, make sure you know the device name of your CD-ROM drive. Ask your system administrator for this information.



On Solaris and IRIX systems with Volume Management, load the CD-ROM and then go to Step 5. (On these systems, the CD-ROM automatically mounts on the `/cdrom` directory. To determine whether you have Volume Management, check to see if the Solaris `vold` daemon or the IRIX `mediad` daemon is running on your system.)

To mount the CD-ROM:

1 Load the CD-ROM into the drive.

2 Log in as root:

```
% su root
```

3 If you do not already have one, create a `cdrom` directory to be the mount point for the CD-ROM drive:

```
# mkdir /cdrom
```

4 Mount the CD-ROM:



▫ On Solaris systems without Volume Management:

```
# /etc/mount -r -F hsfs <cdrom-device-name> /cdrom
```



▫ If your HP-UX system is configured to mount the CD-ROM at `/cdrom`:

```
# /etc/mount /cdrom
```

▫ If your HP-UX system is not configured to mount the CD-ROM at `/cdrom`, use the following command:

```
# /etc/mount -r -F cdfs <cdrom-device-name> /cdrom
```



▫ On IRIX 6.x:

```
# /etc/mount -r -t iso9660 <cdrom-device-name> /CDROM
```



▫ On Tru64 UNIX:

```
# mount -r -t cdfs -o rrip <device_name> /cdrom
```

Note: The following error can occur after an attempt to read the CD-ROM on your Tru64 UNIX Alpha machine:

```
/dev/rz4c on /cdrom: No such device
```

If this error occurs, verify that `/dev/rz4c` is the correct device name. If the name is correct, have your system administrator include option `CDFS` in the system configuration file. Then rebuild the kernel. Refer to `man cdfs(4)` for additional information.

5 To verify that the CD-ROM is mounted, use the `ls` command to list the files:

```
# ls -R /cdrom
```

Ejecting the CD-ROM

After you complete the installation, eject the CD-ROM.



On Solaris with Volume Management, type:

```
% eject cdrom
```

On Solaris without Volume Management, type:

```
% su root
# umount /cdrom
# eject cdrom
# exit
```



On HP-UX and Tru64 UNIX, type:

```
% su root
# umount /cdrom
# exit
```

Press the eject button on the CD-ROM drive.



On IRIX, type:

```
% eject /CDROM
```

Using rs_install commands

The `rs_install` program includes four commands that you can use to rerun specific sections of the `rs_install` program without actually reinstalling any products: `license_setup`, `license_check`, `post_install`, and `options_setup`.

To use these commands, go to the `PurifyPlusFamily.<version>` directory. For example:

```
% cd Rational/releases/PurifyPlusFamily.<version>
% ./license_setup
```

- Use the `license_setup` command to rerun the license setup phase of the installation. Use `license_setup` to import your permanent or TLA license keys and whenever you want to change your licensing information.
- Use the `license_check` command to check your license server and the `.dat` license file to make sure your license information is correct.
- Use the `post_install` command to rerun the post-installation phase of the installation. One of the actions this command performs is to call the `<product>.configure` command; see *Step 3: Post-installation configuration tasks* on page 16.

- Use `options_setup` to modify the list of users allowed to use the Rational Software product. For more information, see *Modifying the list of user names* on page 21.

Using the FLEXlm Software License Manager

The FLEXlm Software License Manager monitors license access, simultaneous usage, idle time, and so on. It includes the following components:

- A vendor daemon named `rational` that dispenses Purify, PureCoverage, and Quantify licenses. The `rational` daemon is used for all licensed Rational Software products. If you have products from other vendors that also use FLEXlm, they will include their own vendor daemons.
- A license manager daemon named `lmgrd` that is used by all licensed products from all vendors that use FLEXlm. The `lmgrd` daemon does not process requests on its own, but forwards requests to the appropriate vendor daemon.
- A Rational license file that specifies your license servers, vendor daemons, and product licenses.

The Rational license file

The Rational `.dat` license file is a text file that in most cases is created when you run the `rs_install` or `license_setup` program. The `.dat` license file is based on data from the `.upd` file that you receive from AccountLink.

The file for startup and evaluation licenses is:

```
Rational/config/Temporary.dat
```

The default file for permanent or TLA licenses is:

```
Rational/config/<server name>.dat
```

Note: For best results, use the Rational license file only for Rational Software product licenses.

The `rs_install` program saves the license path to `<producthome>/lm_license_file`. This is the path that Purify, PureCoverage, and Quantify use to locate the license file. You can

override the location in `.lm_license_file` by setting the `LM_LICENSE_FILE` environment variable. The full path searched is equivalent to `$LM_LICENSE_FILE:'cat.lm_license_file'`.

Verifying that FLEXlm is working

To verify that your FLEXlm License Manager is operational and that the daemons are running, type the following on your license server:

```
% /bin/ps -e | egrep "lmgrd|rational"
```

The output should include lines similar to the following:

```
/bin/ps -e | egrep "lmgrd|rational"  
 351 ? 0:00 rational  
 345 ? 0:01 lmgrd
```

Using FLEXlm commands

The FLEXlm License Manager supports the following commands for system administration:

Use this command:	To:
<code>lmdiag</code>	Diagnose problems when you cannot check out a license
<code>lmdown</code>	Shut down the license and vendor daemons
<code>lmhostid</code>	Report the license manager host ID of a workstation
<code>lmreread</code>	Reread the license file and start new vendor daemons
<code>lmstat</code>	Report status on daemons and feature usage
<code>exinstal</code>	Report on licenses in the license file you specify on the command line

Learning more about FLEXlm

For more information about the FLEXlm Software License Manager, see the *FLEXlm End User Manual* that is included on your Rational Software CD-ROM.

The *FLEXlm End User Manual*, along with answers to frequently asked questions about FLEXlm, is also available at <http://www.globetrotter.com/manual.htm>.

Rational Purify: What it does

Rational® Purify® is the most comprehensive runtime error detection tool available. It checks all the code in your program, including any application, system, and third-party libraries. Purify works with complex software applications, including multi-threaded and multi-process applications.

Purify checks every memory access operation, pinpointing *where* errors occur and providing detailed diagnostic information to help you analyze *why* the errors occur. Among the many errors that Purify helps you locate and understand are:

- Reading or writing beyond the bounds of an array
- Using uninitialized memory
- Reading or writing freed memory
- Reading or writing beyond the stack pointer
- Reading or writing through null pointers
- Leaking memory and file descriptors

With Purify, you can develop clean code from the start, rather than spending valuable time debugging problem code later.

This chapter introduces the basic concepts involved in using Purify. For complete information, see the Purify online help system.

Finding errors in Hello World

This chapter shows you how to use Purify to find memory errors in an example Hello World program. If you run the example yourself, you should expect minor platform-related differences in program output from what is shown here.

Before you begin:

- 1 Create a new working directory. Go to the new directory and copy the `hello_world.c` program and related files from the `<purifyhome>/example` directory. For example:

```
% mkdir /usr/home/chris/pwork
% cd /usr/home/chris/pwork
% cp <purifyhome>/example/hello* .
```

- 2 Examine the code in `hello_world.c`. The version of `hello_world.c` provided with Purify is slightly different from the traditional version.

```
1 /*
2  * Copyright (c) 1992-1997 Rational Software Corp.
3  *
4  * ...
5  * This is a test program used in Purifying Hello World
6  */
7
8
9
10
11
12 #include <stdio.h>
13 #include <malloc.h>
14
15 static char *helloWorld = "Hello, World";
16
17 main()
18 {
19     char *mystr = malloc(strlen(helloWorld));
20
21     strncpy(mystr, helloWorld, 12);
22     printf("%s\n", mystr);
23 }
```

At first glance there are no obvious errors, yet the program actually contains a memory access error and leaked memory that Purify will help you to identify.

Instrumenting a program

- 1 Compile and link the Hello World program, then run the program to verify that it produces the expected output:

```
% cc -g hello_world.c
```

```
% a.out
```

output ————— Hello, World

- 2 Instrument the program by adding `purify` to the front of the compile/link command line. To get the maximum amount of detail in Purify messages, use the `-g` option:

```
% purify cc -g hello_world.c
```



Note: On Tru64 UNIX and IRIX, you can add `purify` in front of the compile/link command line, or you can Purify the executable. On Tru64 UNIX, use the `-taso` option with `purify` if you linked with the `-taso` option:

```
% purify <-taso> a.out
```

You then run the instrumented program by typing:

```
% a.out.pure
```



On Tru64 UNIX and IRIX, Purify caches Dynamic Shared Objects (DSOs), not object files. References to linkers and link-line options in this book do not apply to Purify on Tru64 UNIX or IRIX.

Compiling and linking in separate stages

If you compile and link your program in separate stages, specify `purify` only on the link line. For example:

On the compile line, use:

```
% cc -c -g hello_world.c
```

On the link line, use:

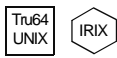
```
% purify cc -g hello_world.o
```

Running the instrumented program

Run the instrumented Hello World program:



```
% a.out
```



On Tru64 UNIX and IRIX, if you use `purify` on the executable instead of `on` on the compile/link line, type:

```
% a.out.pure
```

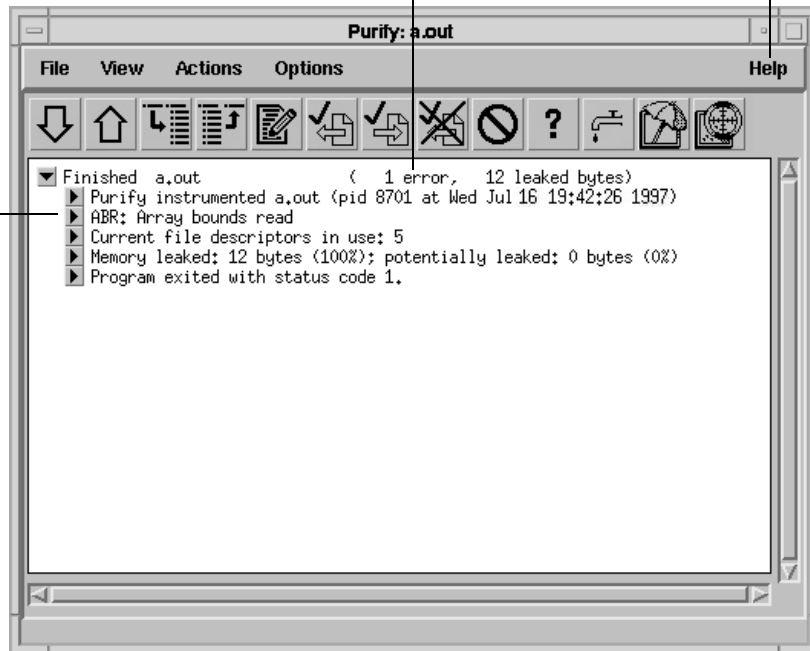
This prints “Hello, World” in the current window and displays the Purify Viewer.

Purify displays the number of access errors and leaked bytes detected

Click for a list of Purify error messages

The Purify Viewer displays messages about the program, including errors such as this ABR error

For a description of a message, right click the message, then select **Explain message** from the pop-up menu

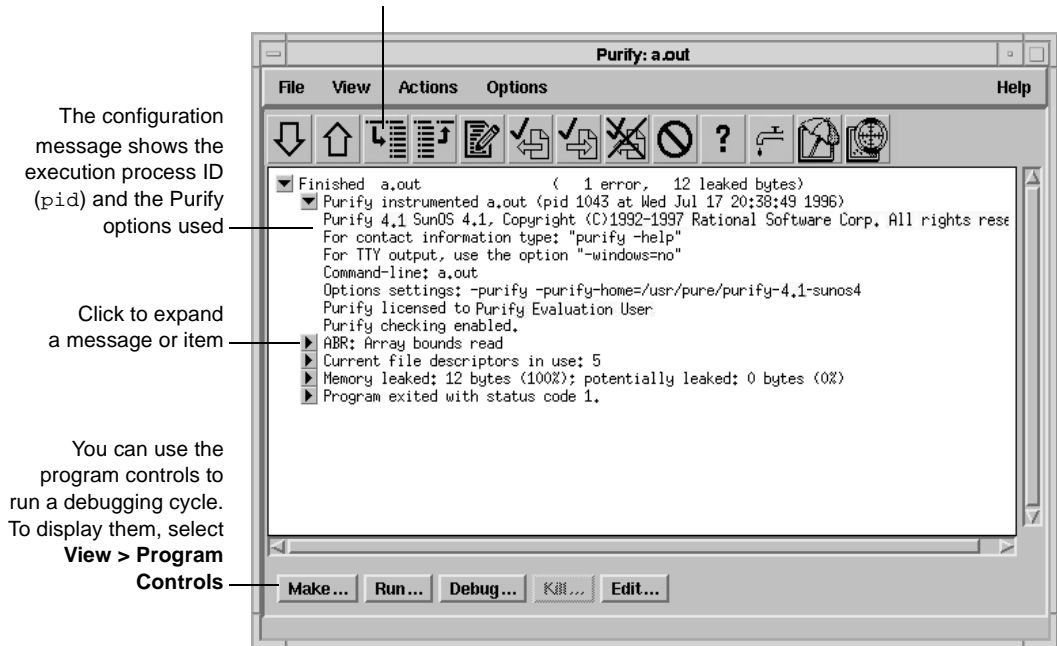


Notice that the instrumented Hello World program starts, runs, and exits normally. Purify does not stop the program when it finds an error.

Seeing all your errors at a glance

The Purify Viewer displays the results of the run of the instrumented Hello World program. You can expand each message to see additional details.

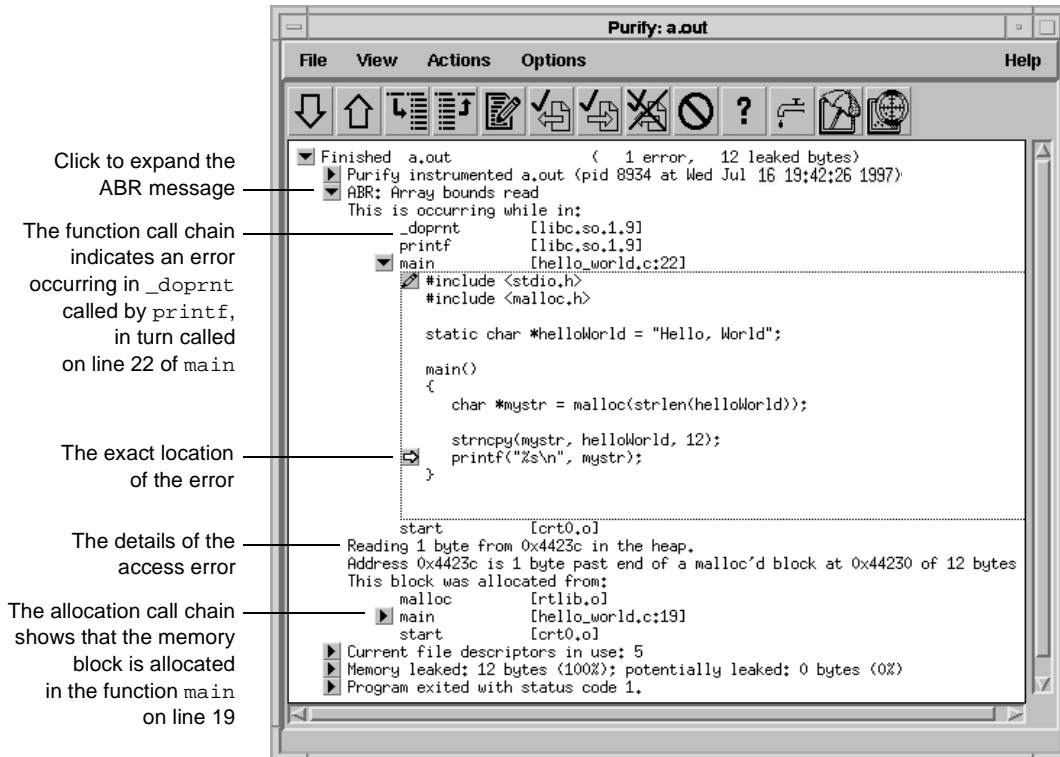
Select one or more messages in the Viewer, then click to expand the messages



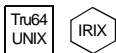
Note: The Viewer displays messages for a single executable only. It is specific to the name of the executable, the directory containing the executable, and the user ID.



Finding and correcting errors

Purify reports an array bounds read (ABR) memory access error in the Hello World program. You can expand the ABR message to see the exact location of the error.



Note: To make debugging easier, Purify reports line numbers, source filenames, and local variable names whenever possible if you use the `-g` compiler option when you instrument the program. If you do not use the `-g` option, Purify reports only function names and object filenames.



On Tru64 UNIX and IRIX, system libraries retain their source file and line number information; therefore, the  can appear next to a system library function whose source file is not available. When you click the  for such a line, Purify prompts you for the location of the source file. Enter the location of the file if you know it, and then click **OK** to expand the line.

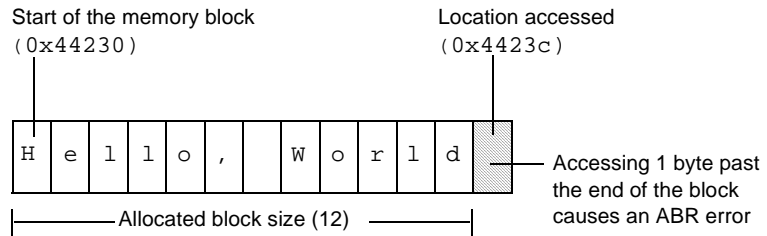
Understanding the cause of the error

To understand the cause of the ABR error, look at the code in `hello_world.c` again.

```
.
.
.
15 static char *helloWorld = "Hello, World";
16
17 main()
18 {
19     char *mystr = malloc(strlen(helloWorld));
20
21     strncpy(mystr, helloWorld, 12);
22     printf("%s\n", mystr);
23 }
```

Purify reports that the ABR error occurs here

On line 22, the program requests `printf` to display `mystr`, which is initialized by `strncpy` on line 21 for the 12 characters in “Hello, World.” However, `_doprnt` is accessing one byte more than it should. It is looking for a `NULL` byte to terminate the string. The extra byte for the string’s `NULL` terminating character has *not* been allocated and initialized.

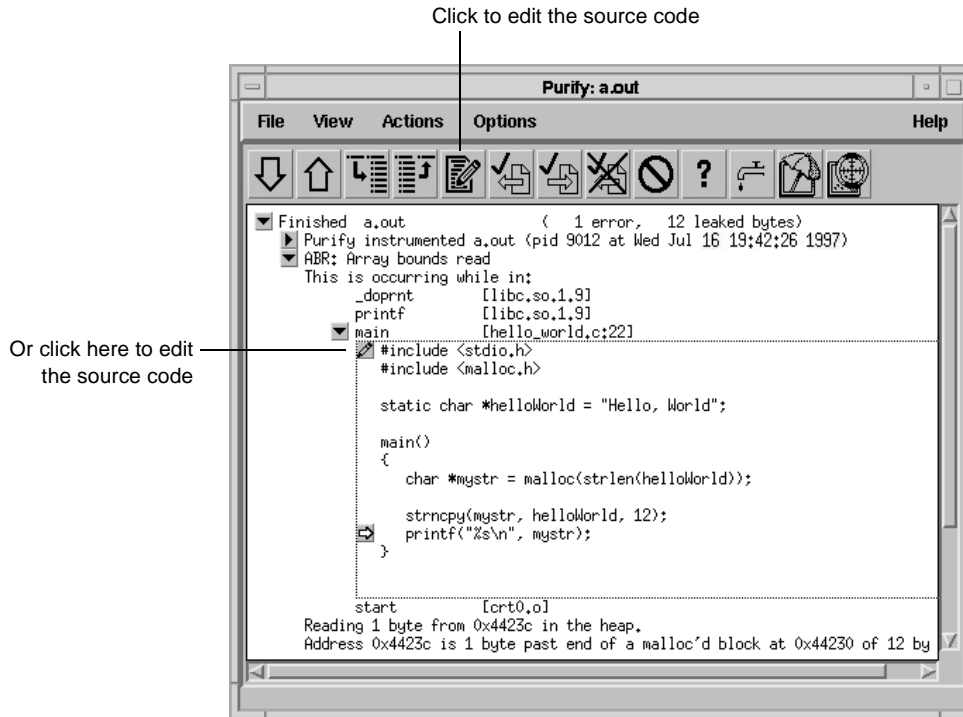


For more information, see *How Purify finds memory-access errors* on page 48.

Correcting the ABR error

To correct this ABR error:

- 1 Click the Edit tool  to open an editor.



Note: By default, Purify displays seven lines of the source code file in the Viewer. You can change the number of lines of source code displayed by setting an X resource.

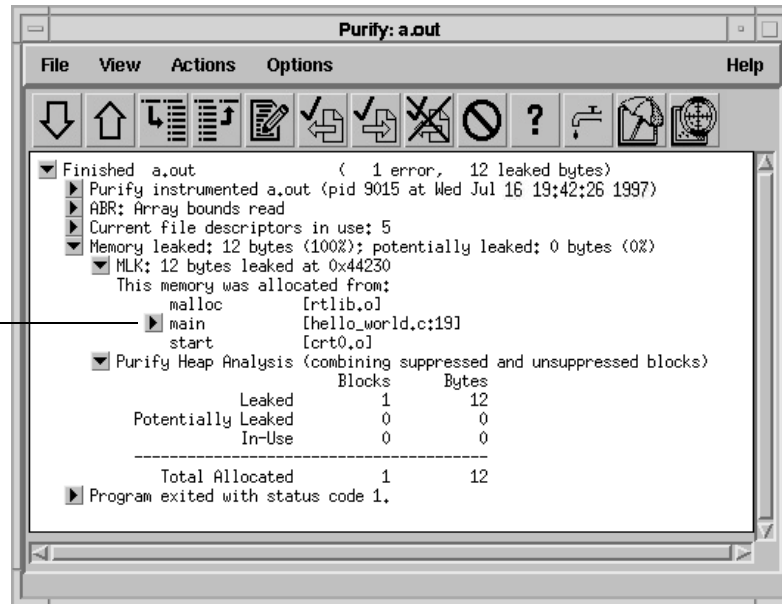
- 2 Change lines 19 and 21 as follows:

```
19 char *mystr = malloc(strlen(helloWorld)+1);
20
21 strncpy(mystr, helloWorld, 13);
```


Correcting the MLK error


It is not immediately obvious why this memory was leaked. If you look closer, however, you can see that this program does not have an `exit` statement at the end. Because of this omission, the `main` function returns rather than calls `exit`, thereby making `mystr`—the only reference to the allocated memory—go out of scope.

Line 19 of `hello_world.c` in `main` allocates 12 bytes of leaked memory. The start of this memory block is `0x44230`, the same block with the array bounds read error in `_doprnt`



If `main` called `exit` at the end, `mystr` would remain in scope at program termination, retaining a valid pointer to the start of the allocated memory block. Purify would then have reported it as memory in use rather than memory leaked. Alternatively, `main` could `free` `mystr` before returning, deallocating the memory so it is no longer in use or leaked.

To correct this MLK error:

- 1 Click the Edit tool  to open an editor.
- 2 Add a call to `exit(0)` at the end of the Hello World program.

Looking at the heap analysis

Purify distinguishes between three memory states, reporting both the number of blocks in each state and the sum of their sizes:

- Leaked memory
- Potentially leaked memory
- Memory in use

A true memory leak (MLK) is memory to which your program has no pointer

A potential memory leak (PLK) is memory that does not have a pointer to its beginning, but does have one to its interior

Memory in use (MIU) is memory to which your program has pointers (these are not leaks)

```
Finished a.out ( 1 error, 12 leaked bytes)
Purify instrumented a.out (pid 9015 at Wed Jul 16 19:42:26 1997)
ABR: Array bounds read
Current file descriptors in use: 5
Memory leaked: 12 bytes (100%); potentially leaked: 0 bytes (0%)
MLK: 12 bytes leaked at 0x44230
This memory was allocated from:
malloc [rtlib.o]
main [hello_world.c:19]
start [crt0.o]
Purify Heap Analysis (combining suppressed and unsuppressed blocks)
-----
                Blocks      Bytes
Potentially Leaked  0          0
In-Use             0          0
-----
Total Allocated    1         12
Program exited with status code 1.
```

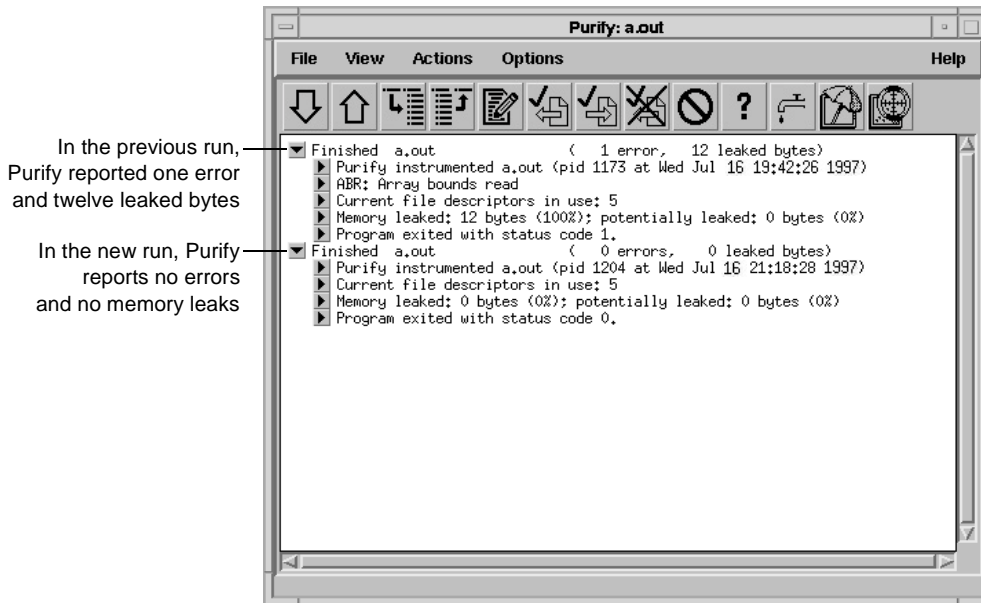
The exit status message provides information about:

- *Basic memory usage containing* statistics not easily available from a single shell command. It includes program code and data size, as well as maximum heap and stack memory usage in bytes.
- *Shared-library memory usage* indicating which libraries were dynamically linked and their sizes.

Comparing program runs

To verify that you have corrected the ABR and MLK errors, recompile the program with `purify`, and run it again.

Purify displays the results of the new run in the same Viewer as the previous run so it's easy to compare them. In this simple Hello World program, you can quickly see that the new run no longer contains the ABR and MLK errors.



Congratulations! You have successfully Purify'd the Hello World program.

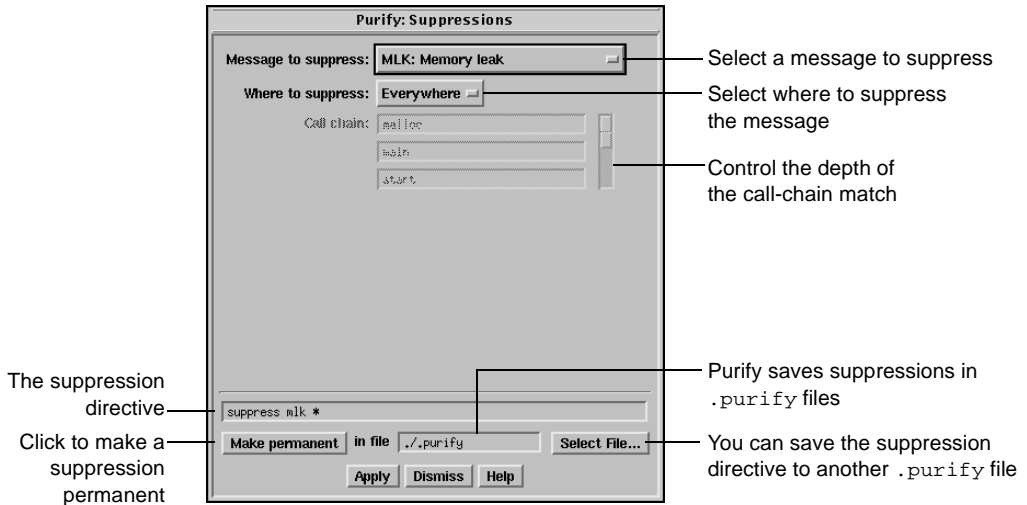
Suppressing Purify messages

A large program can generate hundreds of error messages. To quickly focus on the most critical ones, you can suppress the less critical messages based on their type and source. For example, you might want to hide all informational messages, or hide all messages that originate in a specific file.

You can suppress messages in the Viewer either during or after a run of your program. To suppress a message in the Viewer:

- 1 Select the message you want to suppress.
- 2 Select **Options > Suppressions**.

Purify displays the Suppressions dialog, containing information about the selected message.



You can also specify suppressions directly in a `.purify` file. Suppressions created in the Viewer take precedence over suppressions in `.purify` files; however, they apply only to the current Purify session. Unless you click **Make permanent**, they do not remain when you restart the Viewer.

Saving Purify output to a view file

A view file is a binary representation of all messages generated in a Purify run that you can browse with the Viewer or use to generate reports independent of a Purify run. You can save a run to a view file to compare the results of one run with the results of subsequent runs, or to share the file with other developers.

Saving a run to a view file from the Viewer

To save a program run to a view file from the Viewer:

- 1 Wait until the program finishes running, then click the run to select it.
- 2 Select **File > Save As**.
- 3 Type a filename, using the `.pv` extension to identify the run as a Purify view file.

Opening a view file

To open a view file from the Viewer:

- 1 Select **File > Open**.
- 2 Select the view file you want to open.

Purify displays the run from the view file in the Viewer. You can work with the run just as you would if you had run the program from the Viewer.

You can also use the `-view` option to open a view file. For example:

```
% purify -view <filename>.pv
```

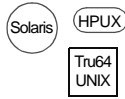
This opens the `<filename>.pv` view file in a new Viewer.

Using your debugger with Purify

You can run an instrumented program directly under your debugger so that when Purify finds an error, you can investigate it immediately.

Alternatively, you can enable Purify's just-in-time (JIT) debugging feature to have Purify start your debugger *only* when it encounters an error—and you can specify which types of errors trigger the debugger. JIT debugging is useful for errors that appear only once in a while. When you enable JIT debugging, Purify suspends execution of your program just before the error occurs, making it easier to analyze the error.

Using Purify with PureCoverage



Purify is designed to work closely with PureCoverage, Rational Software's runtime test coverage tool. PureCoverage identifies the parts of your program that have not yet been tested so you can tell whether you're exercising your program sufficiently for Purify to find all the memory errors in your code.

To use Purify with PureCoverage, add both product names to the front of your link line. Include all ' with the program to which they refer. For example:

```
% purify <purifyoptions> purecov <purecovoptions> \  
cc -g hello_world.c -o hello_world
```

To start PureCoverage from the Purify Viewer, click the PureCoverage icon  in the toolbar.

For more information, see *Using Rational PureCoverage* on page 51.

Purify API functions

You can call Purify's API functions from your source code or from your debugger to gain more control over Purify's error checking. By calling these functions from your debugger, you get additional control without modifying your source code. You can use Purify's API functions to check memory state and to search for memory and file-descriptor leaks.

For example, by default Purify reports memory leaks only when you exit your program. However, if you call the API function `purify_new_leaks` at key points throughout your program, Purify reports the memory leaks that have occurred since the last time the function was called. This periodic checking enables you to locate and track memory leaks more effectively.

To use Purify API functions, include `<purifyhome>/purify.h` in your code and link with `<purifyhome>/purify_stubs.a`.

Commonly used functions	Description
<code>int purify_describe (char *addr)</code>	Prints specific details about memory
<code>int purify_is_running (void)</code>	Returns "TRUE" if the program is instrumented

Commonly used functions	Description
<code>int purify_new_inuse (void)</code>	Prints a message on all memory newly in use
<code>int purify_new_leaks (void)</code>	Prints a message on all new leaks
<code>int purify_new_fds_inuse (void)</code>	Lists the new open file descriptors
<code>int purify_printf (char *format, ...)</code>	Prints formatted text to the Viewer or log-file
<code>int purify_watch (char *addr)</code>	Watches for memory write, malloc, free
<code>int purify_watch_n (char *addr, int size, char *type)</code>	Watches memory: type = "r", "w", "rw"
<code>int purify_watch_info (void)</code>	Lists active watchpoints
<code>int purify_watch_remove (int watchno)</code>	Removes a specified watchpoint
<code>int purify_what_colors (char *addr, int size)</code>	Prints the color coding of memory

Build-time options

Specify build-time options on the link line when you instrument a program with Purify. For example:

```
% purify -cache-dir=$HOME/cache -always-use-cache-dir cc ...
```

Commonly used build-time options	Default
<code>-always-use-cache-dir</code> Forces all instrumented object files to be written to the global cache directory	no
<code>-cache-dir</code> Specifies the global directory where Purify caches instrumented object files	<purifyhome>/cache
<code>-collector</code> Specifies the collect program to handle static constructors (for use with gcc, g++)	none
<code>-ignore-runtime-environment</code> Prevents the runtime Purify environment from overriding the option values used in building the program	no

Commonly used build-time options	Default
-linker Sets the alternative linker to build the executables instead of the system default	system-dependent
-print-home-dir Prints the name of the directory where Purify is installed, then exits	

Conversion characters for filenames

Use these conversion characters when specifying filenames for options such as `-log-file` and `-view-file`.

Character	Converts to
%V	Full pathname of program with "/" replaced by "_"
%v	Program name
%p	Process id (pid)
qualified filenames (./%v.pv)	Absolute or relative to current working directory
unqualified filenames (no '/')	Directory containing the program

Runtime options

Specify runtime options on the link line or by using the `PURIFYOPTIONS` environment variable. For example:

```
% setenv PURIFYOPTIONS "-log-file=mylog.%v.%p" `printenv PURIFYOPTIONS`
```

Commonly used runtime options	Default
-auto-mount-prefix Removes the prefix used by file system auto-mounters	/tmp_mnt
-chain-length Sets the maximum number of stack frames to print in a report	6
-fds-in-use-at-exit Specifies that the file descriptor in use message be displayed at program exit	yes

Commonly used runtime options	Default
-follow-child-processes Controls whether Purify monitors child processes in an instrumented program	no
-jit-debug Enables just-in-time debugging	none
-leaks-at-exit Reports all leaked memory at program exit	yes
† -log-file Writes Purify output to a log file instead of the <i>Viewer</i> window	stderr
-messages Controls display of repeated messages: "first", "all", or in a "batch" at program exit	first
-program-name Specifies the full pathname of the instrumented program if argv[0] contains an undesirable or incorrect value	argv[0]
-show-directory Shows the directory path for each file in the call chain, if the information is available	no
-show-pc Shows the full pc value in each frame of the call chain	no
-show-pc-offset Appends a pc-offset to each function name in the call chain	no
† -view-file Saves Purify output to a view file (.pv) instead of the <i>Viewer</i> .	none
-user-path Specifies a list of directories in which to search for programs and source code	none
-windows Redirects Purify output to stderr instead of the <i>Viewer</i> if -windows=no	none

† Can use the conversion characters listed on page 45.

Purify messages

Purify reports the following messages. For detailed, platform-specific information, see the Purify online help system.

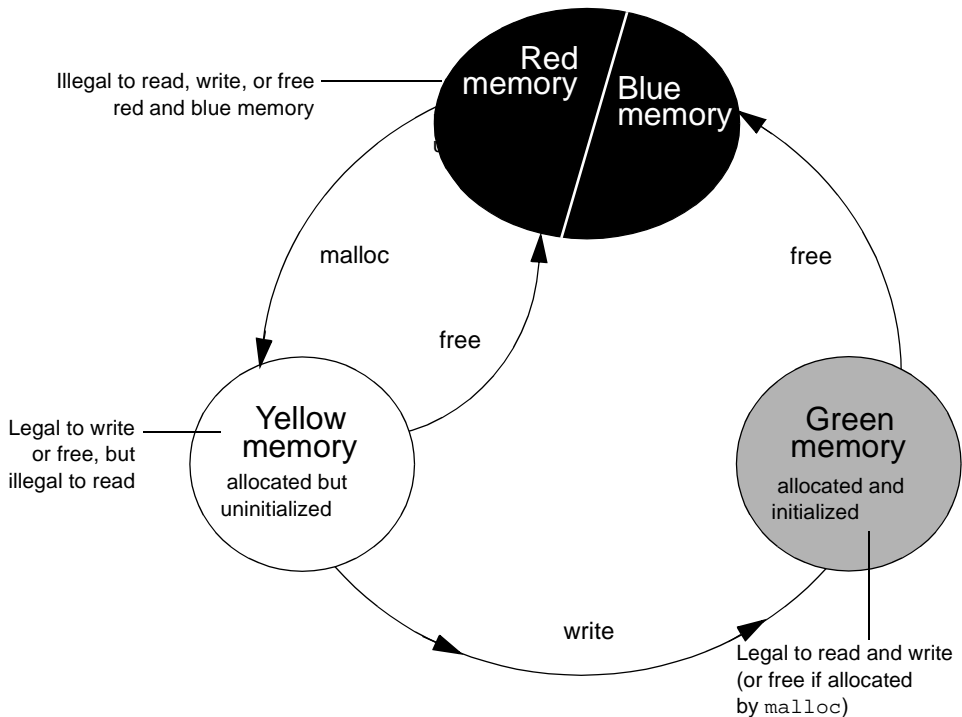
Message	Description	Severity*	Message	Description	Severity*
ABR	Array Bounds Read	W	NPR	Null Pointer Read	F
ABW	Array Bounds Write	C	NPW	Null Pointer Write	F
BRK	Misuse of Brk or Sbrk	C	PAR	Bad Parameter	W
BSR	Beyond Stack Read	W	PLK	Potential Leak	W
BSW	Beyond Stack Write	W	PMR	Partial UMR	W
COR	Core Dump Imminent	F	SBR	Stack Array Bounds Read	W
FIU	File Descriptors In Use	I	SBW	Stack Array Bounds Write	C
FMM	Freeing Mismatched Memory	C	SIG	Signal	I
FMR	Free Memory Read	W	SOF	Stack Overflow	W
FMW	Free Memory Write	C	UMC	Uninitialized Memory Copy	W
FNH	Freeing Non Heap Memory	C	UMR	Uninitialized Memory Read	W
FUM	Freeing Unallocated Memory	C	WPF	Watchpoint Free	I
IPR	Invalid Pointer Read	F	WPM	Watchpoint Malloc	I
IPW	Invalid Pointer Write	F	WPN	Watchpoint Entry	I
MAF	Malloc Failure	I	WPR	Watchpoint Read	I
MIU	Memory In-Use	I	WPW	Watchpoint Write	I
MLK	Memory Leak	W	WPX	Watchpoint Exit	I
MRE	Malloc Reentrancy Error	C	ZPR	Zero Page Read	F
MSE	Memory Segment Error	W	ZPW	Zero Page Write	F

* Message severity: F=Fatal, C=Corrupting, W=Warning, I=Informational

How Purify finds memory-access errors

Purify monitors every memory operation in your program, determining whether it is legal. It keeps track of memory that is not allocated to your program, memory that is allocated but uninitialized, memory that is both allocated and initialized, and memory that has been freed after use but is still initialized.

Purify maintains a table to track the status of each byte of memory used by your program. The table contains two bits that represent each byte of memory. The first bit records whether the corresponding byte has been allocated. The second bit records whether the memory has been initialized. Purify uses these two bits to describe four states of memory: red, yellow, green, and blue.



Purify checks each memory operation against the color state of the memory block to determine whether the operation is valid. If the program accesses memory illegally, Purify reports an error.

- *Red*: Purify labels heap memory and stack memory red initially. This memory is unallocated and uninitialized. Either it has never been allocated, or it has been allocated and subsequently freed.

In addition, Purify inserts guard zones around each allocated block and each statically allocated data item, in order to detect array bounds errors. Purify colors these guard zones red and refers to them as *red zones*. It is illegal to read, write, or free red memory because it is not owned by the program.

- *Yellow*: Memory returned by `malloc` or `new` is yellow. This memory has been allocated, so the program owns it, but it is uninitialized. You can write yellow memory, or free it if it is allocated by `malloc`, but it is illegal to read it because it is uninitialized. Purify sets stack frames to yellow on function entry.
- *Green*: When you write to yellow memory, Purify labels it green. This means that the memory is allocated and initialized. It is legal to read or write green memory, or free it if it was allocated by `malloc` or `new`. Purify initializes the *data* and *bss* sections of memory to green.
- *Blue*: When you free memory after it is initialized and used, Purify labels it blue. This means that the memory is initialized, but is no longer valid for access. It is illegal to read, write, or free blue memory.

Since Purify keeps track of memory at the byte level, it catches all memory-access errors. For example, it reports an uninitialized memory read (UMR) if an `int` or `long` (4 bytes) is read from a location previously initialized by storing a `short` (2 bytes).

How Purify checks statically allocated memory

In addition to detecting access errors in dynamic memory, Purify detects references beyond the boundaries of data in global variables and static variables; that is, data allocated statically at link time as opposed to dynamically at run time.

Here is an example of data that is handled by the static checking feature:

```
int array[10];
main() {
    array[11] = 1;
}
```

In this example, Purify reports an array bounds write (ABW) error at the assignment to `array[11]` because it is 4 bytes beyond the end of the array.

Purify inserts red zones around each variable in your program's static-data area. If the program attempts to read from or write to one of these red zones, Purify reports an array bounds error (ABR or ABW).

Purify inserts red zones into the data section *only* if all data references are to known data variables. If Purify finds a data reference that is relative to the start of the data section as opposed to a known data variable, Purify is unable to determine which variable the reference involves. In this case, Purify inserts red zones at the beginning and end of the data section only, not between data variables.

Purify provides several command-line options and directives to aid in maximizing the benefits of static checking.

Rational PureCoverage: What it does

During the development process, software changes daily, sometimes hourly. Unfortunately, test suites do not always keep pace. Rational® PureCoverage® is a simple, easily deployed tool that identifies the portions of your code that have not been exercised by testing.

Using PureCoverage, you can:

- Identify the portions of your application that your tests have not exercised
- Accumulate coverage data over multiple runs and multiple builds
- Merge data from different programs sharing common source code
- Work closely with Purify to make sure that Purify finds errors throughout your *entire* application
- Automatically generate a wide variety of useful reports
- Access the coverage data so you can write your own reports

PureCoverage provides the information you need to identify gaps in testing quickly, saving precious time and effort.

This chapter introduces the basic concepts involved in using PureCoverage. For complete information, see the PureCoverage online help system.

Finding untested areas of Hello World

This chapter shows you how to use PureCoverage to find the untested parts of the `hello_world.c` program.

Before you begin:

- 1 Create a new working directory. Go to the new directory, and copy the `hello_world.c` program and related files from the `<purecovhome>/example` directory:

```
% mkdir /usr/home/pat/example
% cd /usr/home/pat/example
% cp <purecovhome>/example/hello* .
```

- 2 Examine the code in `hello_world.c`.

The version of `hello_world.c` provided with PureCoverage is slightly more complicated than the usual textbook version.

```
#include <stdio.h>
void display_hello_world();
void display_message();

main(argc, argv)
int argc;
char** argv;
{
    if (argc == 1)
        display_hello_world();
    else
        display_message(argv[1]);
    exit(0);
}

void
display_hello_world()
{
    printf("Hello, World\n");
}

void
display_message(s)
    char *s;
{
    printf("%s, World\n", s);
}
```

Instrumenting a program

- 1 Compile and link the Hello World program, then run the program to verify that it produces the expected output:

```
% cc -g hello_world.c
% a.out
```

output ————— Hello, World

- 2 Instrument the program by adding `purecov` to the front of the compile/link command line. To have PureCoverage report the maximum amount of detail, use the `-g` option:

```
% purecov cc -g hello_world.c
```

Note: If you compile your code *without* the `-g` option, PureCoverage provides only function-level data. It does not show line-level data.

Tru64
UNIX

On Tru64 UNIX, you can add `purecov` in front of the compile/link command line, or you can instrument the executable. Use the `-taso` option with `purecov` if you linked with the `-taso` option:

```
% purecov <-taso> a.out
```

Tru64
UNIX

On Tru64 UNIX, PureCoverage caches Dynamic Shared Objects (DSOs), not object files. References to linkers and link-line options in this chapter do not apply to PureCoverage on Tru64 UNIX.

A message appears, indicating the version of PureCoverage that is instrumenting the program:

```
PureCoverage 4.4 Solaris 2, Copyright 1994-1999 Rational
Software Corp.
All rights reserved.
Instrumenting: hello_world.o Linking
```

Note: When you compile and link in separate stages, add `purecov` only to the link line.

Running the instrumented program

Run the instrumented Hello World program:

```
% a.out
```



On Tru64 UNIX, if you use `purecov` on the executable instead of on the compile/link line, type:

```
% a.out.pure
```

PureCoverage displays the following:

	Name of the instrumented executable	You can use this command to display technical support contact information
Start-up banner	**** PureCoverage instrumented a.out (pid 3466 at Wed Feb 3 10:32:40 1999) * PureCoverage 4.4 Solaris 2, Copyright 1994-1999 Rational Software Corp. * All rights reserved. * For contact information type: "purecov -help" * Command-line: a.out * Options- settings : -purecov \ -purecov-home=/usr/pure/purecov-4.4-solaris2 * PureCoverage licensed to Rational Software Corp. * Coverage counting enabled.	
Normal program output	-----Hello, World	
PureCoverage saves coverage data to a .pcv file	**** PureCoverage instrumented a.out (pid 3466) **** * Saving coverage data to /usr/home/pat/example/a.out.pcv. * To view results type: purecov -view /usr/home/pat/example/a.out.pcv	

The `a.out` program produces its normal output, just as if it were not instrumented. When the program completes execution, PureCoverage writes coverage information for the session to the file `a.out.pcv`. Each time the program runs, PureCoverage updates this file with additional coverage data.

Displaying coverage data

To display the coverage data for the program, use the command:

```
% purecov -view a.out.pcv &
```

This displays the PureCoverage Viewer.

These columns show statistics for function usage

These columns show statistics for line usage

This column shows the number of adjusted lines

Summary information for the entire program

Information for the source directory

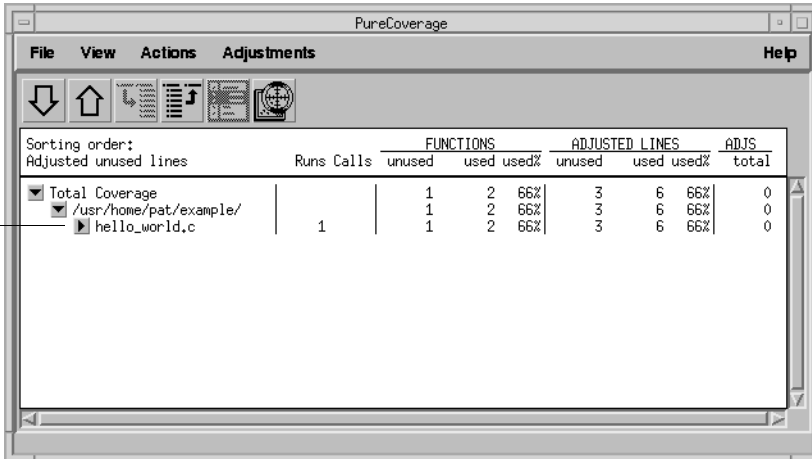
Adjusted unused lines	Runs	Calls	FUNCTIONS			ADJUSTED LINES			ADJS total
			unused	used	used%	unused	used	used%	
<input checked="" type="checkbox"/> Total Coverage			1	2	66%	3	6	66%	0
<input checked="" type="checkbox"/> /usr/home/pat/example/			1	2	66%	3	6	66%	0

In this example, there is only one source directory, so the information displayed for the directory is identical to the `Total Coverage` information.

Note: The default header for line statistics is `ADJUSTED LINES`, not just `LINES`. This is because PureCoverage has an adjustment feature that lets you adjust coverage statistics by excluding specific lines. Under certain circumstances, the adjusted statistics give you a more practical reflection of coverage status than the actual coverage statistics. The `ADJS` column in this example contains zeroes, indicating that it does not include adjustments.

Expanding the file-level detail

Click ► next to `.../example/` to expand the file-level information for the directory.



The screenshot shows the PureCoverage application window with a menu bar (File, View, Actions, Adjustments, Help) and a toolbar. Below the toolbar is a table with the following structure:

Sorting order:		FUNCTIONS			ADJUSTED LINES			ADJS		
Adjusted	unused lines	Runs	Calls	unused	used	used%	unused	used	used%	total
▼	Total Coverage			1	2	66%	3	6	66%	0
▼	/usr/home/pat/example/			1	2	66%	3	6	66%	0
▶	hello_world.c	1		1	2	66%	3	6	66%	0

A callout box on the left side of the window points to the expanded directory entry, containing the text: "File-level information includes the number of runs for which PureCoverage collected data".

You used only one file in the `example` directory to build `a.out`. Therefore the `FUNCTIONS` and `ADJUSTED LINES` information for the file is the same as for the directory. The number 1 in the `Runs` column indicates that you ran the instrumented `a.out` only once.

Note: When you are examining data collected for multiple executables, or for executables that have been rebuilt with some changed files, the number of runs can be different for each file.

Examining function-level detail

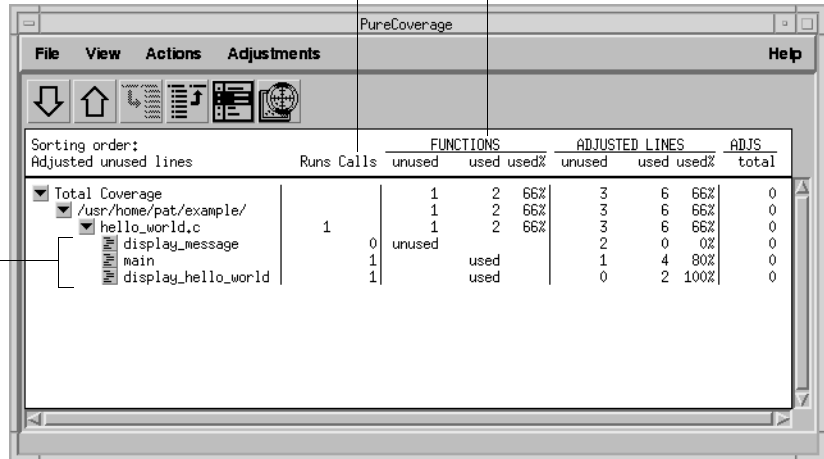
Expand the `hello_world.c` line to show function-level information.

The Viewer shows coverage information for the functions `display_message`, `main`, and `display_hello_world`.

The Calls column shows how many times the program called each function

The FUNCTIONS columns tell at a glance whether each function was used or unused


Function-level information includes the number of times the program called each function

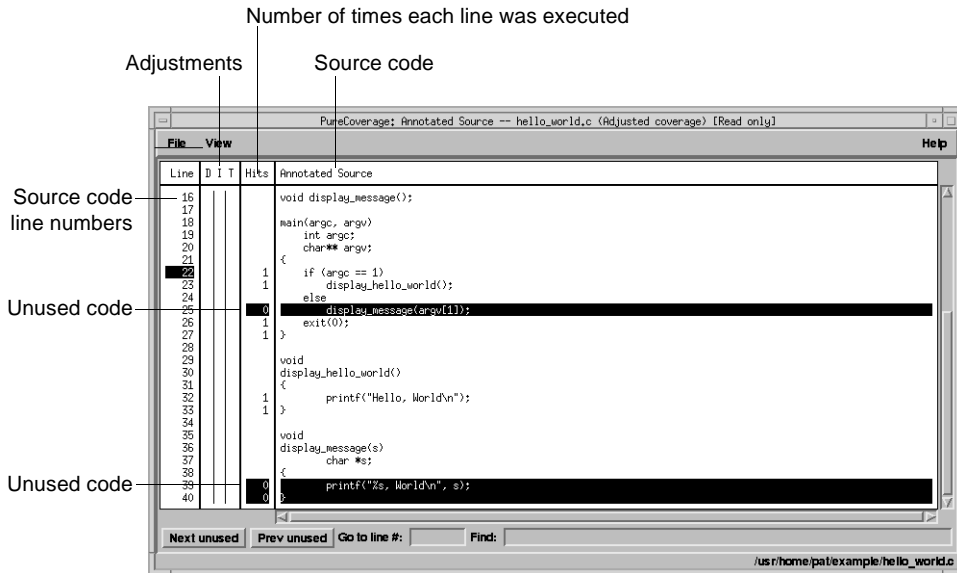


Sorting order: Adjusted unused lines	Runs	Calls	FUNCTIONS			ADJUSTED LINES			ADJS total
			unused	used	used%	unused	used	used%	
▼ Total Coverage			1	2	66%	3	6	66%	0
▼ /usr/home/pat/example/			1	2	66%	3	6	66%	0
▼ hello_world.c	1		1	2	66%	3	6	66%	0
display_message		0	unused			2	0	0%	0
main		1		used		1	4	80%	0
display_hello_world		1		used		0	2	100%	0

PureCoverage does not list the `printf` function or any functions that it calls. The `printf` function is a part of the system library, `libc`. By default, PureCoverage excludes collection of data from system libraries.

Examining the annotated source

To see the source code for `main` annotated with coverage information, click the Annotated Source tool  next to `main` in the Viewer. PureCoverage displays the Annotated Source window.



PureCoverage highlights code that was not used when you ran the program. In this file only two pieces of code were not used:

- The `display_message(argv[1]);` statement in `main`
- The entire `display_message` function

A quick analysis of the code reveals the reason: the program was invoked without arguments.

Improving Hello World's test coverage

To improve the test coverage for Hello World:

- 1 Without exiting PureCoverage, run the program again, this time with an argument. For example:

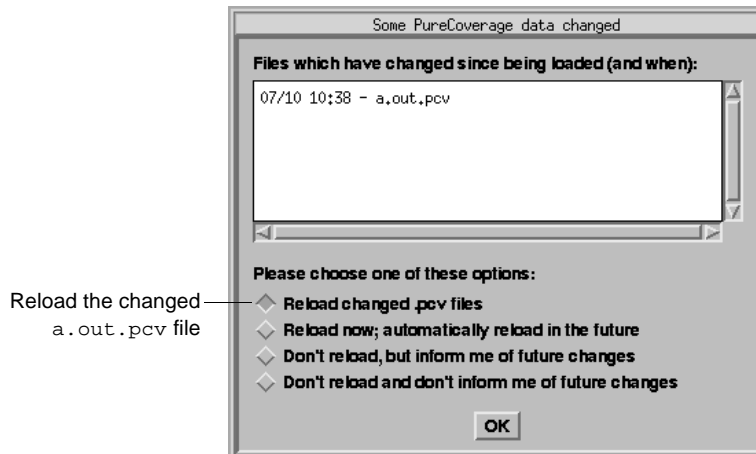
```
% a.out Goodbye
```

PureCoverage displays the following:

```
**** PureCoverage instrumented a.out (pid 17331 at Wed Feb
3 10:38:07 1999) PureCoverage 4.4 Solaris 2, Copyright (C)
1994-1999 Rational Software Corp.
* All rights reserved.
* For contact information type: "purecov -help"
* Command-line: a.out Goodbye
* Options settings: -purecov \
-purecov-home=/usr/pure/purecov-4.4-solaris2
* PureCoverage licensed to Rational Software Corp.
* Coverage counting enabled.
Goodbye, World
```

```
**** PureCoverage instrumented a.out (pid 17331) ****
* Saving coverage data to
/usr/home/pat/example/a.out.pcv.
* To view results type: purecov -view
/usr/home/pat/example/a.out.pcv
```

- 2 PureCoverage displays a dialog confirming that coverage data has changed for this run. Select **Reload changed .pcv files** and click **OK**.



Note: This dialog appears only if the PureCoverage Viewer is open when you run the program.

PureCoverage updates the coverage information in the Viewer and the Annotated Source window.

Function and line coverage is now 100%

The screenshot shows the PureCoverage application window with a menu bar (File, View, Actions, Adjustments, Help) and a toolbar. Below the toolbar is a table with the following data:

Sorting order: Adjusted unused lines	Runs	Calls	FUNCTIONS			ADJUSTED LINES			ADJS total																																													
			unused	used	used%	unused	used	used%																																														
<ul style="list-style-type: none"> ☑ Total Coverage <ul style="list-style-type: none"> /usr/home/pat/example/ <ul style="list-style-type: none"> hello_world.c <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Runs</th> <th>Calls</th> <th>unused</th> <th>used</th> <th>used%</th> <th>unused</th> <th>used</th> <th>used%</th> <th>ADJS total</th> </tr> </thead> <tbody> <tr> <td>2</td> <td></td> <td>0</td> <td>3</td> <td>100%</td> <td>0</td> <td>9</td> <td>100%</td> <td>0</td> </tr> <tr> <td></td> <td></td> <td>1</td> <td></td> <td>used</td> <td>0</td> <td>2</td> <td>100%</td> <td>0</td> </tr> <tr> <td></td> <td></td> <td>1</td> <td></td> <td>used</td> <td>0</td> <td>2</td> <td>100%</td> <td>0</td> </tr> <tr> <td></td> <td>2</td> <td></td> <td></td> <td>used</td> <td>0</td> <td>5</td> <td>100%</td> <td>0</td> </tr> </tbody> </table> 	Runs	Calls	unused	used	used%	unused	used	used%	ADJS total	2		0	3	100%	0	9	100%	0			1		used	0	2	100%	0			1		used	0	2	100%	0		2			used	0	5	100%	0			0	3	100%	0	9	100%	0
Runs	Calls	unused	used	used%	unused	used	used%	ADJS total																																														
2		0	3	100%	0	9	100%	0																																														
		1		used	0	2	100%	0																																														
		1		used	0	2	100%	0																																														
	2			used	0	5	100%	0																																														
			0	3	100%	0	9	100%	0																																													
			1		used	0	2	100%	0																																													
			1		used	0	2	100%	0																																													
	2				used	0	5	100%	0																																													

The statement
display_message
(argv[1]);...
and the function
display_message
are now shown as used

The screenshot shows the PureCoverage application window displaying the annotated source code for hello_world.c. The code is as follows:

```

15 void display_message();
17
18 main(argc, argv)
19 int argc;
20 char** argv;
21 {
22     if (argc == 1)
23         display_hello_world();
24     else
25         display_message(argv[1]);
26     exit(0);
27 }
28
29 void display_hello_world()
30 {
31     printf("Hello, World\n");
32 }
33
34 void display_message(s)
35 char *s;
36 {
37     printf("%s, World\n", s);
38 }
39

```

The code is annotated with 'D I T' and 'Hits' columns. The 'Hits' column shows the number of times each line was executed. The code is annotated as 'used'.

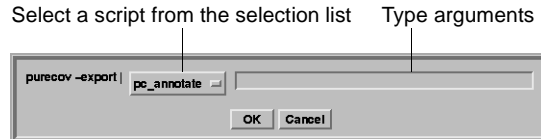
Note: If you still have untested lines, it is possible that your compiler is generating unreachable code.

3 Select File > Exit.

Using report scripts

You can use PureCoverage report scripts to format and process PureCoverage data. The report scripts are located in the `<purecovhome>/scripts` directory.

Select **File > Run script** to open the script dialog.



You can also run report scripts from the command line.

Report scripts

`pc_annotate` Produces an annotated source text file

```
% pc_annotate [-force-merge][--apply-adjustments=no]\
[-file=<basename>...][--type=<type>][<prog>.pcv...]
```

`pc_below` Reports low coverage

```
% pc_below [-force-merge][--apply-adjustments=no][--percent=<pct>]\
[<prog>.pcv...]
```

`pc_build_diff` Compares PureCoverage data from two builds of an application

```
% pc_build_diff [--apply-adjustments=no][--prefix=XXXX...] old.pcv \
new.pcv
```

`pc_covdiff` Annotates the output of diff for modified source code

Note: Cannot run from Viewer

```
% yourdiff <name> | pc_covdiff [--context=<lines>] \
[--format={diff|side-by-side|new-only}][--lines=<boolean>] \
[--tabs=<stops>][--width=<width>][--force-merge][--apply-adjustments=no] \
--file=<name> <prog>.pcv...
```

`pc_diff` Lists files for which coverage has changed

```
% pc_diff [--apply-adjustments=no] old.pcv new.pcv
```

`pc_email` Mails a report to the last person who modified insufficiently covered files

```
% pc_email [--force-merge][--apply-adjustments=no][--percent=<pct>] \
[<prog>.pcv...]
```

`pc_select` Identifies the subset of tests required to exercise modified source code

```
% <list of changed files> | pc_select \
[--diff=<rules>][--canonicalize=<rule>]test1.pcv test2.pcv...
```

Report scripts

`pc_ssheet` Produces a summary in spreadsheet format

```
% pc_ssheet [-force-merge][--apply-adjustments=no][<prog>.pcv...]
```

`pc_summary` Produces an overall summary in table format

```
% pc_summary [-file=<name>...] [--force-merge] [--apply-adjustments=no]
[<prog>.pcv...]
```

Build-time options

You can specify build-time options on the link line when you instrument programs with PureCoverage. For example:

```
% purecov -cache-dir=$HOME/cache --always-use-cache-dir \
cc ...
```

Commonly used build-time options

Default

`--always-use-cache-dir`

no

Forces all PureCoverage instrumented object files to be written to the global cache directory

`--auto-mount-prefix`

/tmp_mnt

Removes the prefix used by file system auto-mounters

`--cache-dir`

<purecovhome>/cache

Specifies the global directory where PureCoverage caches instrumented object files

`--collector`

none

Specifies the collect program to handle static constructors (for use with gcc, g++)

`--ignore-runtime-environment`

no

Prevents the runtime PureCoverage environment from overriding the option values used in building the program

`--linker`

system-dependent

Specifies a linker other than the system default for building the executables

Runtime options

You can specify runtime options on the link line or by using the `PURECOVOPTIONS` environment variable. For example:

```
% setenv PURECOVOPTIONS \  
"-counts-file=./test1.pcv `printenv PURECOVOPTIONS`"
```

	Commonly used runtime options	Default
†	<code>-counts-file</code> Specifies an alternate file for writing coverage count data in binary format	<code>%v.pcv</code>
	<code>-follow-child-processes</code> Controls whether PureCoverage is enabled in forked child processes	<code>no</code>
†	<code>-log-file</code> Specifies a log file for PureCoverage runtime messages	<code>stderr</code>
	<code>-program-name</code> Specifies the full pathname of the PureCoverage instrumented program	<code>argv[0]</code>
†	<code>-user-path</code> Specifies a list of directories to search for source code	<code>none</code>

† Can use the conversion characters listed on page 45.

Analysis-time options

Use analysis-time options with analysis-time mode options. For example:

```
% purecov -merge=result.pcv -force-merge filea.pcv fileb.pcv
```

	Commonly used analysis-time options	Default
	<code>-apply-adjustments</code> Applies all adjustments in the <code>\$HOME/.purecov.adjust</code> file to exported coverage data	<code>yes</code>
	<code>-force-merge</code> Forces the merging of coverage data files (<code>.pcv</code>) obtained from different versions of the same object file	<code>no</code>

Analysis-time mode options

Command-line syntax:

```
% purecov -<mode option> [analysis-time options] \  
<file1.pcv file2.pcv ...>
```

Analysis-time mode options	Compatible options
-export Merges and writes coverage counts from multiple coverage data files (.pcv) in export format to a specified file (-export=<filename>) or to stdout	-apply-adjustments
-extract Extracts adjustment data from source code files and writes it to \$HOME/.purecov.adjust	none
-merge=<filename.pcv> Merges and writes coverage counts from multiple coverage data files (.pcv) in binary format	-force-merge
-view Opens the PureCoverage Viewer for analysis of one or more coverage data files (.pcv)	-force-merge, -user-path

Rational Quantify: What it does

Your application's runtime performance—its speed—is one of its most visible and critical characteristics. Developing high-performance software that meets the expectations of customers is not an easy task. Complex interactions between your code, third-party libraries, the operating system, hardware, networks, and other processes make identifying the causes of slow performance difficult.

Rational® Quantify® is a powerful tool that identifies the portions of your C/C++ or Java application that dominate its execution time. Quantify gives you the insight to quickly eliminate performance problems so that your software runs faster. With Quantify, you can:

- Get accurate and reliable performance data
- Control how data is collected, collecting data for a small portion of your application's execution or the entire run
- Compare *before* and *after* runs to see the impact of your changes on performance
- Easily locate and fix only the problems with the highest potential for improving performance

This chapter introduces the basic concepts involved in using Quantify. For complete information, see the Quantify online help system.

How Quantify works: C/C++

Unlike sampling-based profilers, Quantify reports performance data for your program without any profiler overhead. The numbers you see represent the time your program would take without Quantify. Quantify instruments and reports performance data for *all* the code in your program, including system and third-party libraries, shared libraries, and statically linked modules.

Quantify counts machine cycles: For C/C++ code, Quantify uses Object Code Insertion (OCI) technology to count the instructions your program executes and to compute how many cycles they require to execute. Counting cycles means that the time Quantify records in your code is independent of accidental local conditions and, assuming that the input does not change, identical from run to run. The fact that performance data is **repeatable** enables you to see precisely the effects of algorithm and data-structure changes.

Since Quantify counts cycles, it gives you accurate data at any scale. You do *not* need to create long runs or make numerous short runs to get meaningful data as you must with sampling-based profilers—one short run and you have the data. As soon as you can run a test program, you can collect meaningful performance data and establish a baseline for future comparison.

Quantify times system calls: Quantify measures the elapsed (wall clock) time of each system call made by your program and reports how long your program waited for those calls to complete. You can immediately see the effects of improved file access or reduced network delay on your program. You can optionally choose to measure system calls by the amount of time the kernel records for the process, which is the same as the time the UNIX `/bin/time` utility records.

Quantify distributes time accurately: Quantify distributes each function's time to its callers so you can tell at a glance which function calls were responsible for the majority of your program's time. Unlike `gprof`, Quantify does not make assumptions about the average cost per function. Quantify measures it directly.

How Quantify works: Java

Quantify times performance: Quantify times each method as it executes, and collects accurate data about the actual execution of your Java code. You can choose either to record elapsed wall-clock time or to measure the amount of time the kernel records for the process, like the UNIX `/bin/time` utility. Because data for Java code is based on timing and not counting cycles, as it is for C and C++, performance data for Java code, while reliable for a given run, is not repeatable.

Quantify distributes time accurately: Quantify distributes each method's time to its callers. This helps you detect the methods that are ultimately responsible for bottlenecks in your code.

Collecting performance data: C/C++

To collect performance data for a C/C++ program:

- 1 Add `quantify` to the front of the *link* command line. For example:

```
% quantify cc -g hello_world.c -o hello_world
```

- 2 Run the instrumented program as you usually do:

```
% hello_world
```



Note: On Tru64 UNIX, you can add `quantify` in front of the `compile/link` command line, or you can instrument the executable. Use the `-taso` option with `quantify` if you linked with the `-taso` option:

```
% quantify <-taso> a.out
```

You then run the instrumented program by typing:

```
% a.out.pure
```

Note also that on Tru64 UNIX, Quantify caches Dynamic Shared Objects (DSOs), not object files. References to linkers and link-line options in this chapter do not apply to Quantify on Tru64 UNIX.

When the program starts, Quantify prints license and support information, followed by the expected output from your program.

```
**** Quantify instrumented hello_world (pid 20352 at Sat 5
08:41:27 1999)
Quantify 4.4 Solaris 2, Copyright 1993-1999 Rational
Software Corp.
* For contact information type: "quantify -help"
* Quantify licensed to Quantify Evaluation User
* Quantify instruction counting enabled.
```

Program output—Hello, World.

Data transmission—Quantify: Sending data for 37 of 1324 functions
from hello_world (pid 20352).....done.

When the program finishes execution, Quantify transmits the performance data it collected to `qv`, Quantify's data-analysis program.

Interpreting the program summary: C/C++

After each dataset is transmitted, Quantify prints a program summary showing at a glance how the original, non-instrumented, program is expected to perform.

Time Quantify expects the original program to take

```
Quantify: Resource Statistics for hello_world (pid 20352)
*
*   cycles      secs
* Total counted time:      16148821  0.323 (100.0%)
*   Time in your code:           2721  0.000 ( 0.0%)
*   Time in system calls:      843950  0.017 ( 5.2%)
*   Dynamic library loading:  15302150  0.306 ( 94.8%)
*
* Note: Data collected assuming a sparcstation_lx with clock rate of 750 MHz.
* Note: These times exclude Quantify overhead and possible memory effects.
*
* Elapsed data collection time:      0.336 secs
*
* Note: This measurement includes Quantify overhead.
```

Time spent executing program functions (compute-bound) ————

Time spent waiting for system calls to complete ————

Time spent loading dynamic libraries ————

Time taken to collect data includes Quantify's counting overhead and any memory effects ————

Collecting performance data: Java

To collect Java performance data, run Quantify with the `-java` option, as follows:

- For an applet:

```
% quantify [<Quantify options>] -java <applet viewer>
[<applet viewer options>] <html file>
```

- For a class file:

```
% quantify [<Quantify options>] -java <Java executable>
[<Java options>] <class>
```

- For a JAR file:

```
% quantify [<Quantify options>] -java <Java executable>
[<Java options>] -jar <JAR file>
```

- For a container program such as Netscape Navigator:

```
% quantify [<Quantify options>] -java <exename>
[<arguments to exename>]
```

Note: Quantify can collect line-by-line performance data or method-level data. By default, Quantify uses the line level when debug data, which is stored in class files, is available.

When Quantify starts, it prints license and support information, followed by the expected output from your program.

When the program finishes execution, Quantify transmits the performance data it collected to `qv`, Quantify's data-analysis program.

Interpreting the program summary: Java

After each dataset is transmitted, Quantify prints a program summary showing at a glance how the original, non-instrumented, program is expected to perform.

```

Actual time taken by your process
from when Java was started to the end

Quantify: Resource Statistics for /people/jo/4java/bin/appletviewer (pid 24565)
*
*                               cycles          secs
* Total counted time:          101071710000      134.762 (100.0%)
* Time in your code:           54225003750       72.300 ( 53.7%)
*
*Time Quantify excluded from the dataset:
* Time spent blocked/waiting:  36447111000      48.596
*
* Note: Data collected assuming a UltraSparc with clock rate of 750 MHz.
*
* Note: These measurements performed on a machine with 2 processors.
* For threaded programs on multiprocessors, Quantify will time
* operations that are executed in parallel as if they were performed
* on a single processor.
*

```

Time spent in your Java code, including some Quantify overhead

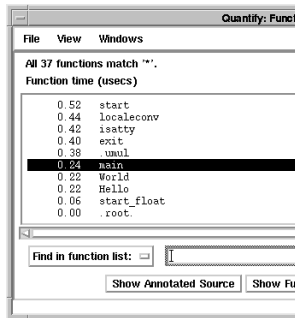
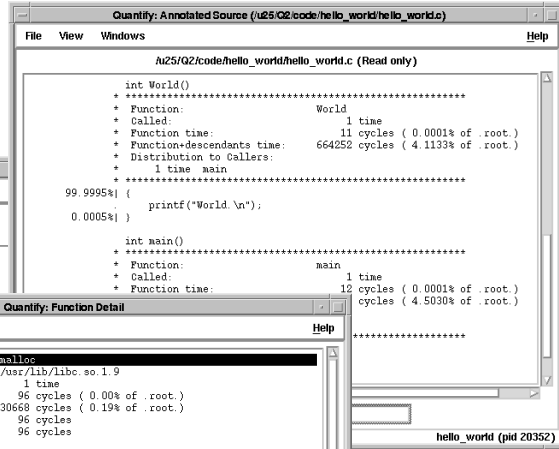
Using Quantify's data analysis windows

After transmitting the last dataset, Quantify displays the Control Panel. From here, you can display Quantify's data analysis windows and begin analyzing your program's performance.

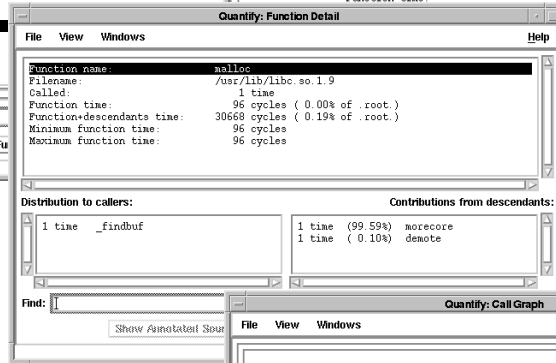
CONTROL PANEL



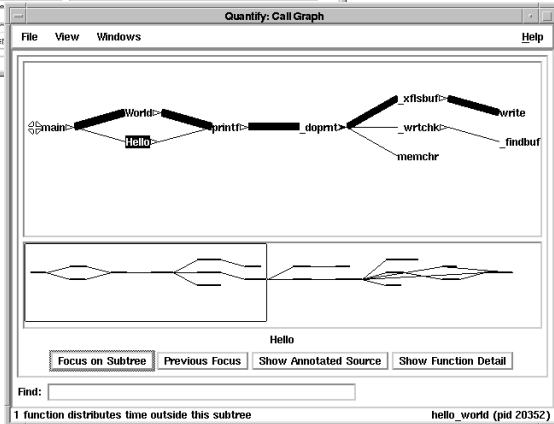
ANNOTATED SOURCE



FUNCTION LIST



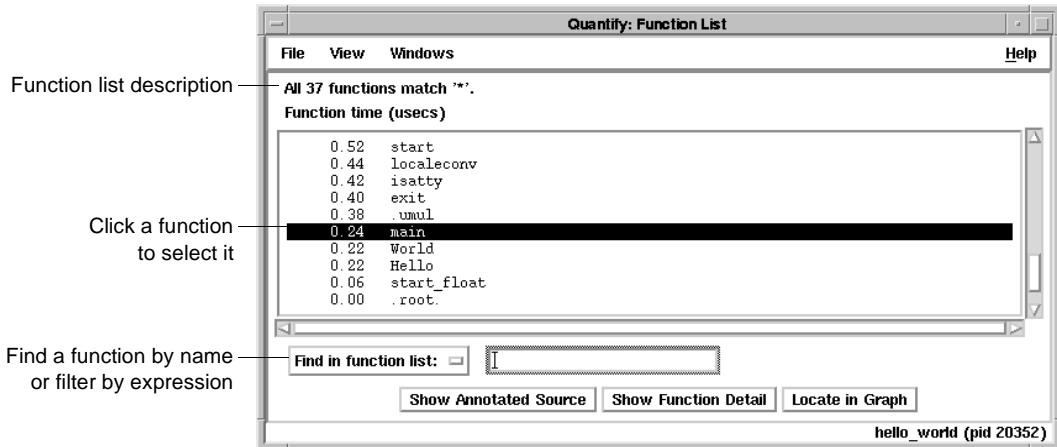
FUNCTION DETAIL



CALL GRAPH

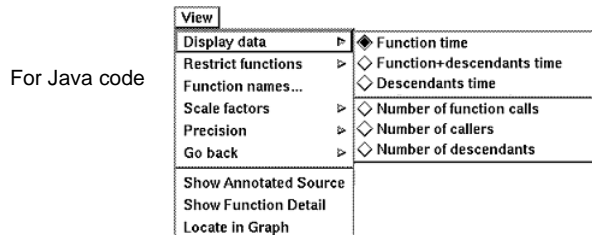
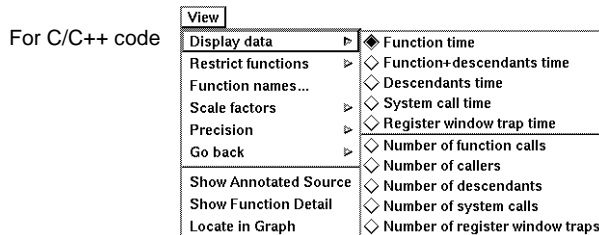
The Function List window

The Function List window shows the functions that your program executed. By default, it displays all the functions in your program, sorted by their *function time*. *This is the amount of time a function spent performing computations (compute-bound) or waiting for system calls to complete.*



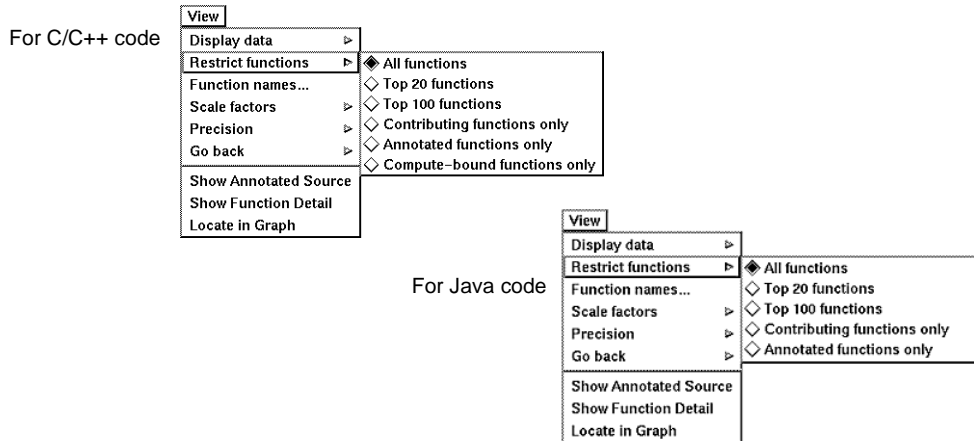
Sorting the function list

To sort the function list based on the various data Quantify collects, select **View > Display data**.



Restricting functions

To focus attention on specific types of functions, or to speed up the preparation of the function list report in large programs, you can restrict the functions shown in the report. Select **View > Restrict functions**.

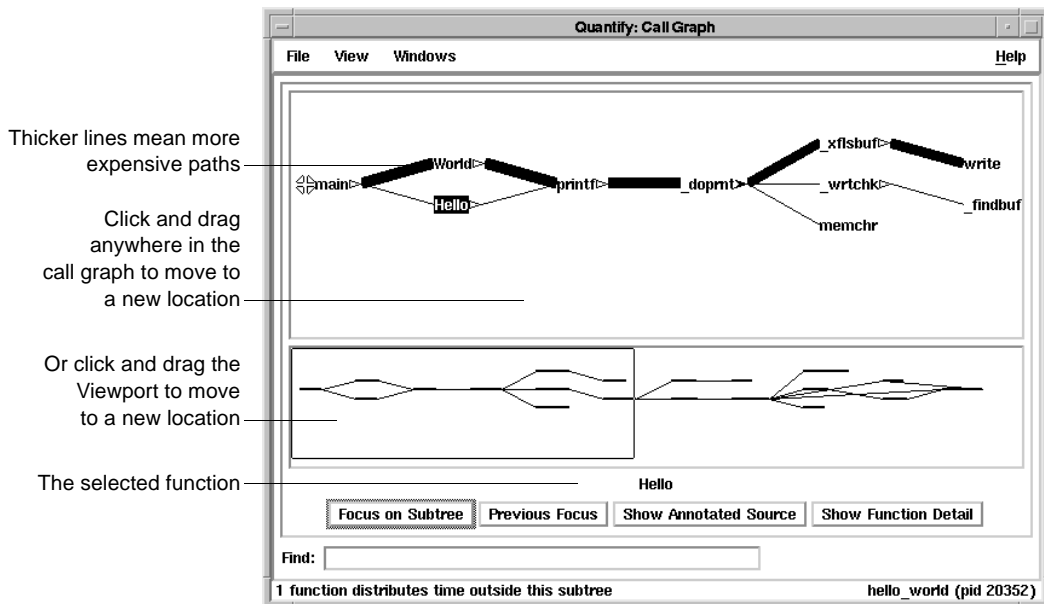


You can restrict the list to the top 20 or top 100 functions in the list, to the functions that have annotated source, to functions that are compute-bound (make no system calls), or to functions that contribute non-zero time for a recorded data type.

The Call Graph window

The Call Graph window presents a graph of the functions called during the run. It uses lines of varying thickness to graphically depict where your program spends its time. Thicker lines correspond directly to larger amounts of time spent along a path.

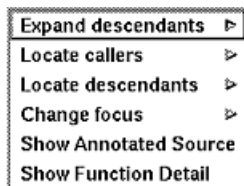
The call graph helps you understand the calling structure of your program and the major call paths that contributed to the total time of the run. Using the call graph, you can quickly discover the sources of bottlenecks.



By default, Quantify expands the call paths to the top 20 functions contributing to the overall time of the program.

Using the pop-up menu

To display the pop-up menu, right-click any function in the call graph.

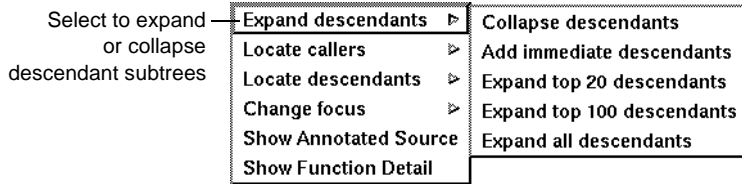


You can use the pop-up menu to:

- Expand and collapse the function's subtree
- Locate individual caller and descendant functions
- Change the focus of the call graph to the selected function
- Display the annotated source code or the function detail for the selected function

Expanding and collapsing descendants

Use the pop-up menu to expand or collapse the subtrees of descendants for individual functions.



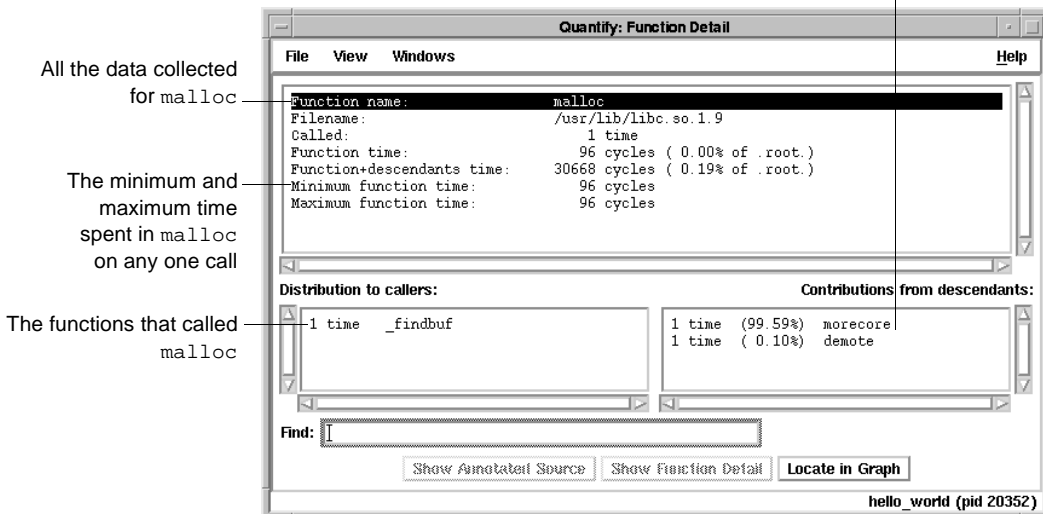
After expanding or collapsing subtrees, you can select **View > Redo layout** to remove any gaps that your changes create in the call graph.

The Function Detail window

The Function Detail window presents detailed performance data for a single function, showing its contribution to the overall execution of the program.

For each function, Quantify reports both the time spent in the function's own code (its *function* time) and the time spent in all the functions that it called (its *descendants* time). Quantify distributes this accumulated *function+descendants* time to the function's immediate caller.

The immediate descendants of `malloc`, and how they contributed to `malloc`'s function+descendants time

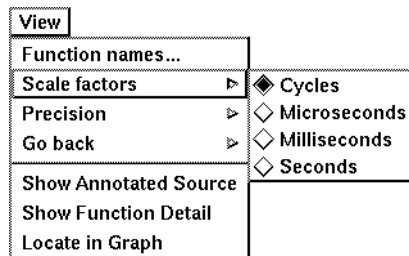


Double-click a caller or descendant function to display the detail for that function.

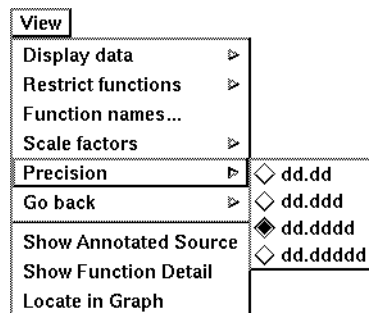
The function time and the function+descendants time are shown as a percentage of the total accumulated time for the entire run. These percentages help you understand how this function's computation contributed to the overall time of the run. These times correspond to the thickness of the lines in the call graph.

Changing the scale and precision of data

Quantify can display the recorded data in cycles (the number of machine cycles) and in microseconds, milliseconds, or seconds. To change the scale of data, select **View > Scale factors**.



To change the precision of data, select **View > Precision**.



Saving function detail data

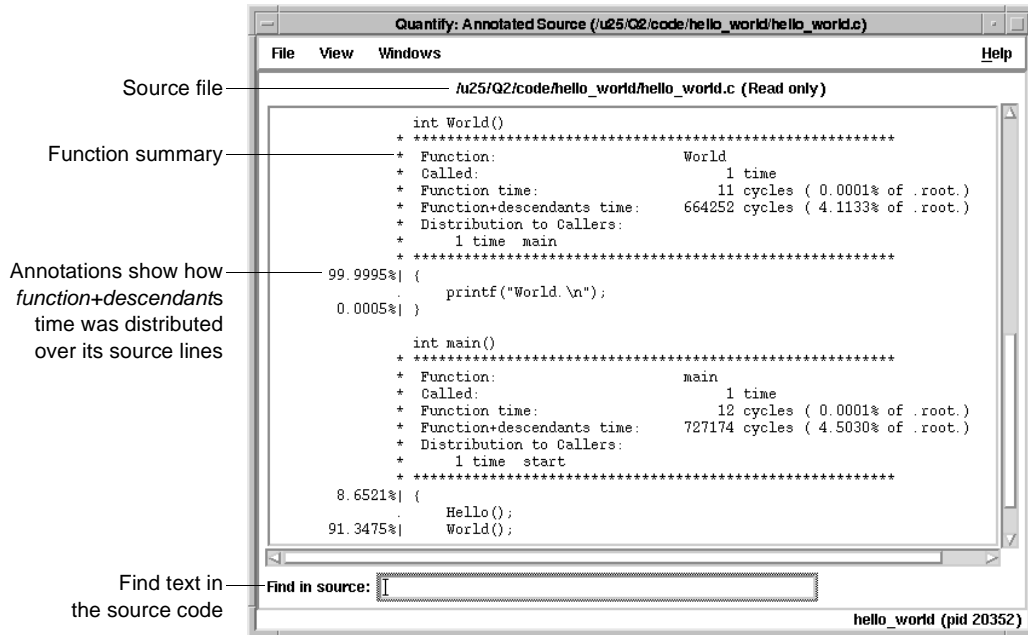
To save the current function detail display to a file, select **File > Save current function detail as**.

To append additional function detail displays to the same file, select **File > Append to current detail file**.

The Annotated Source window

Quantify's Annotated Source window presents line-by-line performance data using the function's source code.

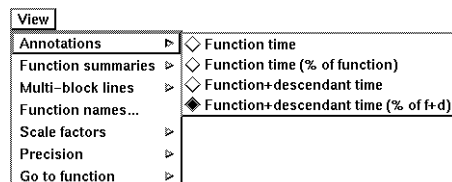
Note: The Annotated Source window is available only for files that you compile using the `-g` debugging option.



The numeric annotations in the margin reflect the time recorded for that line or basic block over all calls to the function. By default, Quantify shows the function time for each line, scaled as a percentage of the total function time accumulated by the function.

Changing annotations

To change annotations, use the View menu. You can select both *function* and *function+descendants* data, either in cycles or seconds and as a percentage of the *function+descendants* time.



Saving performance data on exit

To exit Quantify, select **File > Exit Quantify**. If you analyze a dataset interactively, Quantify does not automatically save the last dataset it receives. When you exit, you can save the dataset for future analysis.



By default, Quantify names dataset files to reflect the program name and its runtime process identifier. You can analyze a saved dataset at a later time by running `qv`, Quantify's data analysis program.

You can also save Quantify data in export format. This is a clear-text version of the data suitable for processing by scripts.

Comparing program runs with `qxdiff`

The `qxdiff` script compares two export data files and reports any changes in performance. For C or C++ programs, the results show exactly how much your program's performance has improved. For Java code, the results indicate general performance trends. This is because C and C++ performance data, based on counting cycles, is repeatable, while Java data, based on the timing of methods, is not repeatable.

To use the `qxdiff` script:

- 1 Save baseline performance data to an export file. Select **File > Export Data As** in any data analysis window.
- 2 Change the program and run Quantify on it again.
- 3 Select **File > Export Data As** to export the performance data for the new run.
- 4 Use the `qxdiff` script to compare the two export data files. For example:

```
% qxdiff -i testHash.pure.20790.0.qx  
improved_testHash.pure.20854.0.qx
```

You can use the `-i` option to ignore functions that make calls to system calls.

Below is the output from this example.

```
Differences between:
program testHash.pure (pid 20790) and
program improved_testHash.pure (pid 20854)

qxdiff lists the
functions that have
changed ...
and summarizes the
differences for the
entire run
```

Function name	Calls	Cycles	% change
strcmp	-40822	-1198640	93.77% faster
putHash	0	-32912	6.61% faster
getHash	0	-28376	7.86% faster
remHash	0	-7856	5.91% faster
hashIndex	0	10000	1.49% slower

5 differences; -1257784 cycles (-0.025 secs at 50 MHz)
25.01% faster overall (ignoring system calls).

Build-time options

Specify build-time options on the link line when you instrument a program with Quantify. For example:

```
% quantify -cache-dir=$HOME/cache -always-use-cache-dir \  
cc ...
```

Commonly used build-time options	Default
-always-use-cache-dir Specifies whether instrumented files are written to the global cache directory	no
-cache-dir Specifies the global cache directory	<quantifyhome>/cache
-collection-granularity Specifies the level of collection granularity	line
† -collector Specifies the collect program to handle static constructors in C++ code	none
† -ignore-runtime-environment Prevents the runtime Quantify environment from overriding option values used in building the program	no

	Commonly used build-time options	Default
†	-linker Specifies an alternative linker to use instead of the system linker	system-dependent
†	-use-machine Specifies the build-time analysis of instruction times according to a particular machine	system-dependent

† Does not apply to Java.

qv runtime options

To run `qv`, specify the option and the saved `.qv` file. For example:

```
% qv -write-summary-file a.out.23.qv
```

	qv options	Default
	-add-annotation Specifies a string to add to the binary file	none
	-print-annotations Writes the annotations to <code>stdout</code>	no
	-windows Controls whether Quantify runs with the graphical interface	yes
	-write-export-file Writes the recorded data in the dataset to a file in export format	none
	-write-summary-file Writes the program summary for the dataset to a file	none

Runtime options

Specify runtime options on the link line or by using the `QUANTIFYOPTIONS` environment variable. For example:

```
% setenv QUANTIFYOPTIONS "-windows=no"; a.out
```

	Commonly used runtime options	Default
†	-avoid-recording-system-calls Avoids recording specified system calls	system-dependent
	-measure-timed-calls Specifies measurement for timing system calls	elapsed-time
†	-record-child-process-data Records data for child processes created by <code>fork</code> and <code>vfork</code>	no
†	-record-system-calls Records system calls	yes
†	-report-excluded-time Reports time that was excluded from the dataset	0.5
	-run-at-exit Specifies a shell script to run when the program exits	none
	-run-at-save Specifies a shell script to run each time the program saves counts	none
	-save-data-on-signals Saves data on fatal signals	yes
	-save-thread-data Saves composite or per-stack thread data	composite
	-write-export-file Writes the dataset to an export file as ASCII text	none
	-write-summary-file Writes the program summary for the dataset to a file	/dev/tty
	-windows Specifies whether Quantify runs with the graphical interface	yes

† Does not apply to Java.

API functions: C/C++

To use Quantify API functions with C/C++ code, include `<quantifyhome>/quantify.h` in your code and link with `<quantifyhome>/quantify_stubs.a`

Commonly used C/C++ functions	Description
<code>quantify_help (void)</code>	Prints description of Quantify API functions
<code>quantify_is_running (void)</code>	Returns <code>true</code> if the executable is instrumented
<code>quantify_print_recording_state (void)</code>	Prints the recording state of the process
<code>quantify_save_data (void)</code>	Saves data from the start of the program or since last call to <code>quantify_clear_data</code>
<code>quantify_save_data_to_file (char * filename)</code>	Saves data to a file you specify
<code>quantify_add_annotation (char * annotation)</code>	Adds the specified string to the next saved dataset
<code>quantify_clear_data (void)</code>	Clears the performance data recorded to this point
† <code>quantify_<action>_recording_data (void)</code>	Starts and stops recording of all data
† <code>quantify_<action>_recording_dynamic_library_data (void)</code>	Starts and stops recording dynamic library data
† <code>quantify_<action>_recording_register_window_traps (void)</code>	Starts and stops recording register-window-trap data
† <code>quantify_<action>_recording_system_call (char *system_call_string)</code>	Starts and stops recording specific system-call data
† <code>quantify_<action>_recording_system_calls (void)</code>	Starts and stops recording of all system-call data

† `<action>` is one of: `start`, `stop`, `is`. For example:
`quantify_stop_recording_system_call`

API methods: Java

You can call an API method from your Java code or from a debugger. Use the following syntax:

```
Rational.PureAPI.IsRunning()
```

or

```
import Rational.PureAPI;
...
PureAPI.IsRunning()
```

PureAPI is a Java class that includes all the Quantify API methods that can be used with Java code. The PureAPI class is part of a Java package called Rational.jar, which is located in <quantifyhome>.

You can run class files that include calls to PureAPI methods with or without Quantify:

- When you run these class files with Quantify, Quantify automatically sets CLASSPATH and LD_LIBRARY_PATH to access Rational.jar and libQProfJ.so.
- When you run the class files without Quantify, you must add <quantifyhome>/lib32 to your LD_LIBRARY_PATH. In addition, if you do not have a Rational.jar file in your <javahome>/jre/lib/ext directory, you must add <quantifyhome> to your CLASSPATH.

The Java API methods are as follows:

Java API methods: class PureAPI	Description
<code>public static int IsRunning();</code>	Returns true if the executable is instrumented
<code>public static int DisableRecordingData();</code>	Disables collection of all data by Quantify
<code>public static int StartRecordingData();</code>	Tells Quantify to start recording all program performance data
<code>public static int StopRecordingData();</code>	Tells Quantify to stop recording all program performance data
<code>public static int IsRecordingData();</code>	Checks if Quantify is currently recording all program performance data

Java API methods: class PureAPI	Description
<code>public static int ClearData();</code>	Tells Quantify to clear all the data it has recorded about your program's performance to this point
<code>public static int SaveData();</code>	Saves all the data recorded since program start (or the last call to <code>clearData()</code>) into a dataset (a <code>.qv</code> file)
<code>public static int AddAnnotation(String annotation);</code>	Tells Quantify to save the argument string in the next output datafile written by <code>saveData()</code>

Index

Symbols

%V, %v, %p 45

A

ABR, array bounds read error
 correcting 36
 in Hello World 34
access errors, how Purify finds 48
account number, Rational Software 3
AccountLink user input 3
-add-annotation 79
adjusted lines 55
-always-use-cache-dir 44, 62, 78
analysis-time options 63
Annotated Source window
 PureCoverage 58
 Quantify 76
a.out.pcv 54
API functions
 Purify 43
 Quantify (C/C++) 81
 Quantify (Java) 82
appending function detail 75
applets, collecting performance data 68
-apply-adjustments 63
-auto-mount-prefix 45, 62
-avoid-recording-system-calls 80

B

blue memory color 49
build-time options
 PureCoverage 62
 Purify 44
 Quantify 78

C

cache directory
 configuring 17
 location of 24
-cache-dir 17, 44, 62, 78
caching dynamic shared objects
 IRIX 31
 Tru64 UNIX 31, 53
caching dynamic shared objects on Tru64
 UNIX 67
caching options
 PureCoverage 62
 Purify 44
 Quantify 78
Call Graph window, Quantify 74
 windows
 Quantify call graph 72
Calls column, PureCoverage 57
CD-ROMs
 ejecting 25
 mounting 24
-chain-length 45
changing annotations, Quantify 76
characters, conversion 45
class files, collecting performance data 68
ClearQuest, integrating 17
code, see source code
collapsing subtrees 74
-collection-granularity 78
-collector 44, 62, 78
color, see memory color
comparing program runs
 with PureCoverage 58
 with Purify 40
 with Quantify qxdiff script 77
compiling and linking 31
compute-bound
 functions 71, 72
 time 68
configuration message 33

- configure command 26
- configuring the cache directory 17
- container programs, collecting Java performance data 69
- controls, Purify program 33
- conversion characters for filenames 45
- counts-file 63
- coverage data
 - file level 56
 - function level 57
 - in PureCoverage Viewer 55
- cycles
 - counted by Quantify 66
 - scale factor 75

D

- daemons, and licensing 27
- .dat license file 6, 9, 27
- data
 - comparing export files 77
 - saving Quantify data 77
- debugger(s)
 - JIT debugging 42
 - scripts on HP-UX 19
 - using with Purify 42
- debugging option, see -g debugging option
- defaults file 15
- deleting product releases 22
- directories
 - cache 17
 - installation 5, 8, 11, 12, 23
 - PureLA 6, 9
 - Rational 23–24
- disk space requirements 4
- dynamic library, timing 68
- dynamic shared object (DSO) caching 31, 53, 67

E

- editing source code 36, 38
- ejecting CD-ROMs 25

- environment variables
 - LM_LICENSE_FILE 28
 - MANPATH 18
 - PATH 18
 - PURECOVOPTIONS 63
 - PUREOPTIONS 17
 - PURIFYOPTIONS 45
 - QUANTIFYOPTIONS 79
- executables, instrumenting
 - PureCoverage on Tru64 UNIX 53
 - Purify on IRIX and Tru64 UNIX 31
 - Quantify on Tru64 UNIX 67
- expanding subtrees 74
- expiration date, licenses 12
- export 64
- exporting Quantify data 77
- extract 64

F

- fds-in-use-at-exit 45
- file(s)
 - a.out.pcv 54
 - installing product 24
 - license_for_*.upd 4
 - Purify view 41
 - qv and qx script 19
 - Rational license (.dat) 6, 9, 27
 - rational.opt 20
 - rs_install.defaults 15
 - Temporary.dat 27
 - users.purela 6, 9
- filename conversion characters 45
- filesystems, installing on read-only 17
- FLEXlm
 - commands 28
 - End User Manual 28
 - GLOBEtroutter web site 28
 - License Manager 27
- floating license 3
- follow-child-processes 46, 63
- force-merge 63

- Function Detail window 74
 - saving data 75
 - scale and precision of data 75
- Function List window
 - finding top contributors 71
 - restricting functions 72
- function+descendants time 74
- functions
 - compute-bound 72
 - coverage detail 57
 - restricting display in Quantify 72
 - sorting in Quantify 71
 - See also API functions
- Functions columns, PureCoverage 57

G

- g debugging option
 - and PureCoverage 53
 - and Purify 34
 - and Quantify 67, 76
- get_hostinfo.sh 3
- GLOBEtrotter web site 28
- graph, see Call Graph window
- green memory color 49

H

- heap analysis, Purify 39
- Hello World example
 - PureCoverage 52
 - Purify 30
- help system, setting up 17
- help, technical support viii
- hiding
 - functions in Quantify 72
 - messages in Purify 41
- hostid for license server host 3
- HP-UX debugger scripts 19

I

- ignore-runtime-environment 44, 62, 78

- installation
 - basic steps 1
 - directory 5, 8, 11, 12, 23
 - evaluation license 2
 - on read-only filesystems 17
 - permanent license 23
 - requirements 4
 - rs_install commands 26
 - startup license 15
 - user input (AccountLink) 3
- instrumenting a program
 - description of vii
 - with PureCoverage 53
 - with Purify 31
 - with Quantify 67
- integration, Purify and PureCoverage 43
- IRIX
 - compile/link command 31
 - DSO caching 31
 - running a Purify instrumented program 32

J

- JAR files, collecting performance data 68
- java 68
- Java and Quantify 66, 68
- jit-debug 46
- just-in-time debugging 42

K

- keys, product license 15

L

- leaks, see memory leaks
- leaks-at-exit 46
- library
 - system and PureCoverage 57
 - time loading dynamic 68
- license daemon, lmgrd 27

- license file
 - .dat 6, 9, 21, 27
 - .upd 4
- license key types 15
- License Manager, FLEXlm 27
- license server
 - port number 8
 - requirements 3
 - using multiple servers 15
- license(s)
 - checking 26
 - expiration date 12
 - floating 3
 - license key types 15
 - named user 6, 9, 16, 18
 - permanent 22, 23
 - quantity 3
 - Rational license file (.dat) 27
 - setting up 26
 - startup 15, 22
 - upd license file 4
 - user names 6, 9, 20–21
- license_check command 26
- license_for_*.upd file 4
- license_setup command 26
- line numbers
 - g option 31, 34
 - on IRIX 34
 - on Tru64 UNIX 34
- linker 45, 62, 79
- links, symbolic 18
- LM_LICENSE_FILE environment variable 28
- lmgrd license daemon 27
- local variable names, displaying 31
- log-file 46, 63

M

- machine cycles 66
- MANPATH environment variable 18
- manual pages 18
- measure-timed-calls 80

- memory access errors
 - example 34
 - how Purify finds 48
- memory color 48
- memory in use message 39
- memory leaks
 - definition 39
 - heap analysis 39
 - message 37
 - new leaks button 37
 - potential 39
 - purify_new_leaks 43
- menu, Quantify pop-up 73
- merge 64
- messages 46
- messages
 - Purify 47
 - suppressing Purify 41
- MLK, memory leak 38
 - example 37
- mounting CD-ROMs 24

N

- Named User license 6, 9, 16, 18, 20
- new memory leaks, Purify 37

O

- Object Code Insertion (OCI) 66
- online help system, setting up 17
- operating system, identifying 24
- options
 - PureCoverage analysis-time 63
 - PureCoverage build-time 62
 - PureCoverage runtime 63
 - Purify build-time 44
 - Purify runtime 45
 - Quantify build-time 78
 - Quantify runtime 79
 - qv runtime 79
- options (by name)
 - add-annotation 79
 - always-use-cache-dir 44, 62, 78

options (by name), *continued*

- apply-adjustments 63
- auto-mount-prefix 45, 62
- avoid-recording-system-calls 80
- cache-dir 44, 62, 78
- chain-length 45
- collection-granularity 78
- collector 44, 62, 78
- counts-file 63
- export 64
- extract 64
- fds-in-use-at-exit 45
- follow-child-processes 46, 63
- force-merge 63
- ignore-run-time-environment 62
- ignore-runtime-environment 44, 78
- java 68
- jit-debug 46
- leaks-at-exit 46
- linker 45, 62, 79
- log-file 46, 63
- measure-timed-calls 80
- merge 64
- messages 46
- print-annotations 79
- print-home-dir 45
- program-name 46, 63
- record-child-process-data 80
- record-system-calls 80
- report-excluded-time 80
- run-at-exit 80
- run-at-save 80
- save-data-on-signals 80
- save-thread-data 80
- show-directory 46
- show-pc 46
- show-pc-offset 46
- taso 31, 53, 67
- use-machine 79
- user-path 46, 63
- view 55, 64
- view-file 46
- windows 46, 79, 80
- write-export-file 79, 80

- write-summary-file 79, 80
- options file 20
- options_setup command 27
- overhead, Quantify 68

P

- PATH environment variable 18
- performance data 67, 69
 - saving 77
- permanent licenses
 - installing manually 23
 - requesting 22
- pop-up menu, Quantify 73
- port number, license server 8
- post_install command 26
- post-installation configuration tasks 16
- potential memory leak 39
- print-annotations 79
- print-home-dir 45
- product license keys 15
- producthome directory 23
- products, removing 22
- program controls, Purify 33
- program runs, comparing
 - Quantify qxdiff script 77
 - with PureCoverage 58
 - with Purify 40
- program summary, Quantify 68, 69
- program-name 46, 63
- programs, running instrumented
 - PureCoverage 54
 - Purify 32
 - Quantify 67
- purecov.configure command 26
- PureCoverage
 - benefits 51
 - symbolic links for 19
 - using with Purify 43
 - Viewer 55
- PURECOVOPTIONS environment variable 63
- PureLA directory 6, 9
- PUREOPTIONS environment variable 17

Purify
 API functions 43
 instrumenting a program 31
 messages 47
 Viewer 32
purify.configure command 26
PURIFYOPTIONS environment variable 45
PurifyPlus vii, 1

Q

Quantify
 API functions (C/C++) 81
 API functions (Java) 82
 build-time options 78
 Call Graph window 72, 74
 overhead 68
 repeatability of timing 66
 runtime options 79
 symbolic links for 19
 with Java 66, 68
quantify.configure command 26
QUANTIFYOPTIONS environment variable 79
qv 67, 69
qv script files 19
qx script files 19
qxdiff script 77

R

Rational ClearQuest, integrating 17
rational daemon 27
Rational license file
 *.dat 27
 license_for_*.upd 4
Rational PurifyPlus vii, 1
Rational Software account number 3
Rational Software website
 AccountLink 4
 home page viii
 technical support viii
rational.opt options file 20
README file location 14
read-only filesystems 17

-record-child-process-data 80
-record-system-calls 80
red memory color 49
Redo layout, Quantify 74
removing previous releases 22
report(s)
 program summary 68, 69
 PureCoverage scripts 61
-report-excluded-time 80
restricting functions in Quantify 72
rs_install
 commands 26
 defaults file 15
 program 14
-run-at-exit 80
-run-at-save 80
running an instrumented program
 PureCoverage 54
 Purify 32
 Quantify 67
runs
 column, PureCoverage 56
 comparing with PureCoverage 58
 comparing with Purify 40
 comparing with Quantify 77
runtime options
 PureCoverage 63
 Purify 45
 Quantify 79
 qv 79

S

-save-data-on-signals 80
-save-thread-data 80
saving
 function detail data 75
 Purify run 41
 Quantify data 77
scale factors 75
scripts
 enabling PureCoverage scripts 19
 HP-UX debugger 19
 PureCoverage report scripts 61

- Quantify 19
- qxdiff 77
- server-name.dat file 27
- servers, license 14
 - requirements 3
 - using multiple 15
- show-directory 46
- show-pc 46
- show-pc-offset 46
- sorting function list 71
- source code
 - annotated in PureCoverage 58
 - annotated in Quantify 76
 - displaying filenames 34
 - editing from Viewer 36, 38
 - line numbers, Purify 34
 - number of lines displayed 36
- startup license 15, 22
- statically allocated memory 49
- subtrees, Quantify 74
- summary, Quantify program 68, 69
- support, technical viii
- suppressing Purify messages 41
- symbolic links 18
 - for HP-UX debugger scripts 19
 - for PureCoverage 19
 - for Purify 18
 - for Quantify 19
- system call timing 66
- system libraries and PureCoverage 57

T

- taso option
 - PureCoverage 53
 - Purify 31
 - Quantify 67
- technical support viii
- Temporary .dat file 27
- time
 - compute-bound 68
 - function+descendants 74

- in code 68
- loading dynamic libraries 68
 - to collect the data 68
- Total Coverage row, PureCoverage 55
- Tru64 UNIX
 - compile/link command 31, 53, 67
 - DSO caching 53, 67
 - running an instrumented program (PureCoverage) 54
 - running an instrumented program (Purify) 31

U

- uname command 24
- uninstall command 22
- .upd license file 4
- use-machine 79
- user names, for licensing 6, 9, 20–21
- user-path 46, 63

V

- validating setup 17
- variable, see environment variable
- view 55, 64
- view file, Purify 41, 42
- Viewer
 - PureCoverage 55
 - Purify 32
- view-file 46
- viewport, call graph 73

W

- websites
 - for obtaining Rational licenses 4
 - GLOBEtrrotter 28
 - Rational software viii
 - Rational technical support viii
- windows 46, 79, 80

windows

- PureCoverage annotated source 58
- PureCoverage viewer 55
- Purify viewer 32
- Quantify annotated source 76
- Quantify call graph 74
- Quantify data analysis 70

Quantify function detail 74

Quantify function list 71

-write-export-file 79, 80

-write-summary-file 79, 80

Y

yellow memory color 49