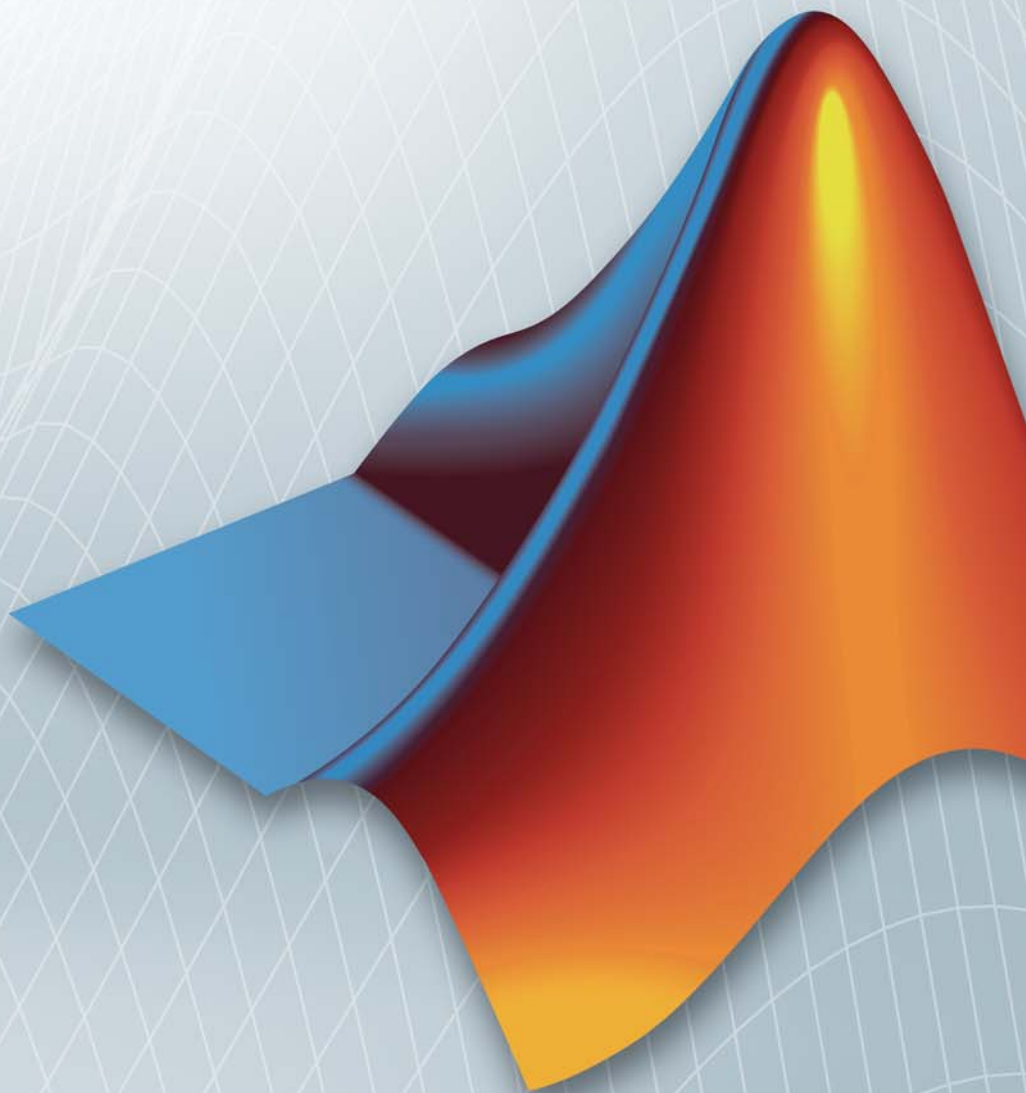


Polyspace® Products for Ada Getting Started Guide

R2012b



How to Contact MathWorks



www.mathworks.com Web
comp.soft-sys.matlab Newsgroup
www.mathworks.com/contact_TS.html Technical Support



suggest@mathworks.com Product enhancement suggestions
bugs@mathworks.com Bug reports
doc@mathworks.com Documentation error reports
service@mathworks.com Order status, license renewals, passcodes
info@mathworks.com Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Polyspace® Products for Ada Getting Started Guide

© COPYRIGHT 1997–2012 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

March 2008	First printing	Revised for Version 5.1 (Release 2008a)
October 2008	Second printing	Revised for Version 5.2 (Release 2008b)
March 2009	Third printing	Revised for Version 5.3 (Release 2009a)
September 2009	Online only	Revised for Version 5.4 (Release 2009b)
March 2010	Online only	Revised for Version 5.5 (Release 2010a)
September 2010	Online only	Revised for Version 6.0 (Release 2010b)
April 2011	Fourth printing	Revised for Version 6.1 (Release 2011a)
September 2011	Online only	Revised for Version 6.2 (Release 2011b)
March 2012	Online only	Revised for Version 6.3 (Release 2012a)
September 2012	Online only	Revised for Version 6.4 (Release 2012b)

Introduction to Polyspace Products for Verifying Ada Code

1

Product Overview	1-2
Polyspace Client for Ada	1-2
Polyspace Server for Ada	1-2
Polyspace Verification	1-4
Overview of Polyspace Verification	1-4
The Value of Polyspace Verification	1-4
Product Components	1-7
Polyspace Verification Environment	1-7
Other Polyspace Components	1-10
Working with Polyspace Software	1-11
Basic Workflow	1-11
The Workflow in This Guide	1-12
Learning More	1-13
Product Help	1-13
MathWorks Online	1-13
Related Products	1-14
Polyspace Products for Verifying C/C++ Code	1-14
Polyspace Products for Linking to Models	1-14

Setting Up Polyspace Project

2

Setting Up a Polyspace Project	2-2
---	-----

About This Tutorial	2-2
Creating a New Project	2-2

Running Verification

3

Running a Verification	3-2
About This Tutorial	3-2
Preparing for Verification	3-3
Starting Server Verification from Project Manager	3-4
Starting Client Verification from Project Manager	3-12

Reviewing Verification Results

4

Reviewing Verification Results	4-2
About This Tutorial	4-2
Opening Verification Results	4-3
Exploring Results Manager Perspective	4-4
Reviewing Results	4-7
Reviewing Results Systematically	4-17
Generating Reports of Verification Results	4-22

Index

Introduction to Polyspace Products for Verifying Ada Code

- “Product Overview” on page 1-2
- “Polyspace Verification” on page 1-4
- “Product Components” on page 1-7
- “Working with Polyspace Software” on page 1-11
- “Learning More” on page 1-13
- “Related Products” on page 1-14

Product Overview

In this section...
“Polyspace Client for Ada” on page 1-2
“Polyspace Server for Ada” on page 1-2

Polyspace Client for Ada

Prove the absence of run-time errors in source code

Polyspace® Client™ for Ada provides code verification that proves the absence of overflow, divide-by-zero, out-of-bounds array access, and certain other run-time errors in source code using static code analysis that does not require program execution, code instrumentation, or test cases. Polyspace Client for Ada uses formal methods-based abstract interpretation techniques to verify code. You can use it on handwritten code, generated code, or a combination of the two, before compilation and test.

Key Features

- Verification of individual packages and package sets
- Formal method based abstract interpretation
- Display of run-time errors directly in code
- Eclipse™ IDE integration

Polyspace Server for Ada

Perform code verification on computer clusters and publish metrics

Polyspace Server™ for Ada provides code verification that proves the absence of overflow, divide-by-zero, out-of-bounds array access, and certain other run-time errors in source code. For faster performance, Polyspace Server for Ada lets you schedule verification tasks to run on a computer cluster. Jobs are submitted to the server using Polyspace Client for Ada. You can integrate jobs into automated build processes and set up e-mail notifications. You can view defects and regressions via a Web browser. You then use the client to download and visualize verification results.

Key Features

- Web-based dashboard providing code metrics and quality status
- Automated job scheduling and e-mail notification
- Multi-server job queue manager
- Verification report generation
- Mixed operating system environment support

Polyspace Verification

In this section...
“Overview of Polyspace Verification” on page 1-4
“The Value of Polyspace Verification” on page 1-4

Overview of Polyspace Verification

Polyspace products verify C, C++, and Ada code by detecting run-time errors before code is compiled and executed.

To verify the source code, you set up verification parameters in a project, run the verification, and review the results. A graphical user interface helps you to efficiently review verification results. Results are color-coded:

- **Green** – Indicates code that never has an error.
- **Red** – Indicates code that always has an error.
- **Gray** – Indicates unreachable code.
- **Orange** – Indicates unproven code (code that might have an error).

The color-coding helps you to quickly identify errors and find the exact location of an error in the source code. After you fix errors, you can easily run the verification again.

The Value of Polyspace Verification

Polyspace verification can help you to:

- “Enhance Software Reliability” on page 1-4
- “Decrease Development Time” on page 1-5
- “Improve Development Process” on page 1-6

Enhance Software Reliability

Polyspace software enhances the reliability of your Ada applications by proving code correctness and identifying run-time errors. Using advanced

verification techniques, Polyspace software performs an exhaustive verification of your source code.

Because Polyspace software verifies all possible executions of your code, it can identify code that:

- Never has an error
- Always has an error
- Is unreachable
- Might have an error

With this information, you can be confident that you know how much of your code is run-time error free, and you can improve the reliability of your code by fixing the errors.

Decrease Development Time

Polyspace software reduces development time by automating the verification process and helping you to efficiently review verification results. You can use it at any point in the development process. However, using it early in coding phases allows you to find errors when they are less costly to fix.

You use Polyspace software to verify Ada source code before compile time. To verify the source code, you set up verification parameters in a project, run the verification, and review the results. This process takes significantly less time than using manual methods or tools that require you to modify code or run test cases.

Color-coding of results helps you to quickly identify errors. You will spend less time debugging because you can see the exact location of an error in the source code. After you fix errors, you can easily run the verification again.

Using Polyspace verification software helps you to use your time effectively. Because you know the parts of your code that do not have errors, you can focus on the code with proven or potential errors.

Reviewing the code that might have errors (orange code) can be time-consuming, but Polyspace software helps you with the review process.

You can use filters to focus on certain types of errors or you can allow the software to identify the code that you should review.

Improve Development Process

Polyspace software makes it easy to share verification parameters and results, allowing the development team to work together to improve product reliability. Once verification parameters have been set up, developers can reuse them for other packages in the same application.

Polyspace verification software supports code verification throughout the development process:

- An individual developer can find and fix run-time errors during the initial coding phase.
- Quality assurance can check overall reliability of an application.
- Managers can monitor application reliability by generating reports from the verification results.

Product Components

In this section...
“Polyspace Verification Environment” on page 1-7
“Other Polyspace Components” on page 1-10

Polyspace Verification Environment

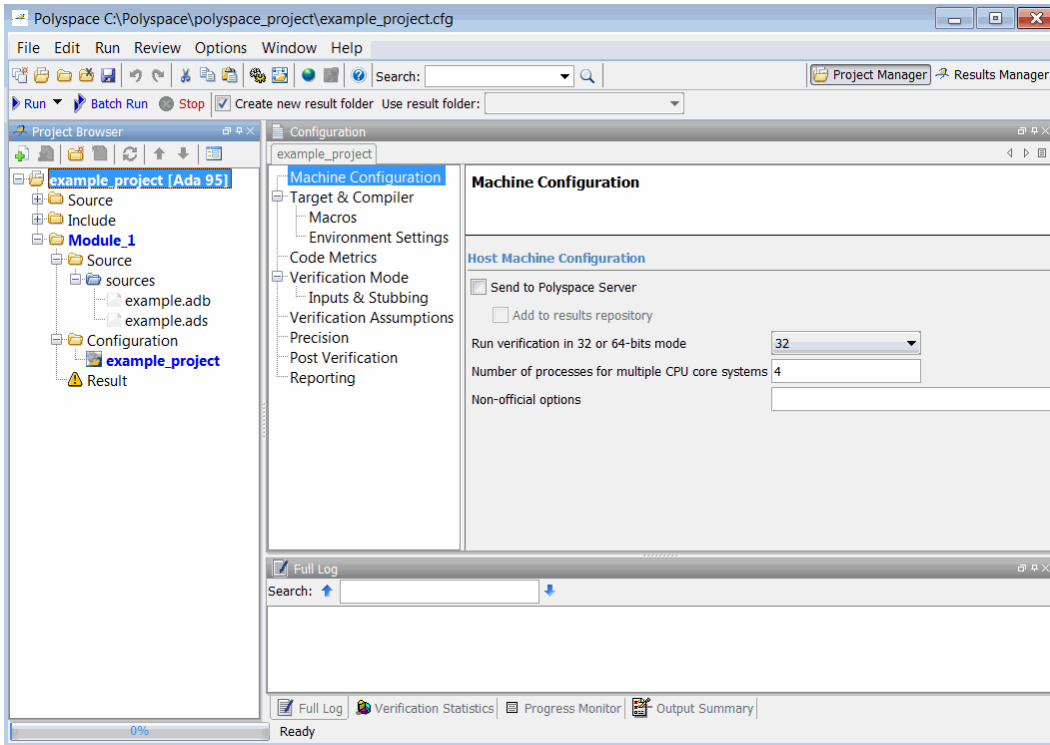
The Polyspace verification environment (PVE) is the graphical user interface of the Polyspace Client for Ada software. You use the Polyspace verification environment to create Polyspace projects, launch verifications, and review verification results.

For Ada verification, you use two perspectives of the Polyspace verification environment:

- “Project Manager Perspective” on page 1-7
- “Results Manager Perspective” on page 1-8

Project Manager Perspective

The Project Manager perspective allows you to create projects, set verification parameters, and start verifications.



You use the Project Manager perspective in the tutorial in “Setting Up a Polyspace Project” on page 2-2.

Results Manager Perspective

The Results Manager perspective allows you to review verification results, comment individual checks, and track review progress.

The screenshot displays the Polyspace IDE interface for a project named "Polyspace - Demo_Ada". The main window is divided into several panes:

- Results Explorer (Left):** Shows a tree view of procedural entities. Under "PKDATA", "ARRAY_OVERFLOW_INIT" is expanded, listing various checks (NIVL.0 to NIVL.28, OVFL.0 to OVFL.15, ZDV.17, OVFL.18 to OVFL.19, OVFL.20, NIVL.22, NIVL.28, OVFL.27) with their respective line numbers and status (e.g., 9, 0, 8, 193).
- Check Details (Top Center):** Displays details for a specific error: "73 ConvToInteger(Res,z/Float(t3(K))); -- OVFL: 'out of bound array acc...". Below this, it shows the error message: "OVFL.27 Error : operation [conversion from -2³¹..2³¹-1 to 1..10] on scalar overflows (results is always...)".
- Source code (Bottom Center):** Shows the Ada source code for "array.ada". The code includes a loop for "I" from 1 to 10, with nested loops for "J" and "K". It performs operations like "t1(I) := I;", "t2(I) := I+1;", "K := K+1;", "t3(I) := t1(J) + t2(J);", and "z := z/float(t3(I));".
- Review Statistics (Top Right):** A table showing the status of various coding review items:

Coding review...	Count	Progr...
Red Scalar OV...	0/1	0
Red justified / ...	0/9	0
Gray justified ...	0/6	0
Orange justifi...	0/8	0
Software rela...	193/225	85
- Call Hierarchy (Right):** Shows the call stack for the current execution point, including "PKDATA.ARRAY_OVERFLOW_INIT", "RANDOM.RANDOM", "PKDATA.CONVTOINTEGER", and "RUNTIME_ERROR.MAINRTE".
- Variable Access (Right):** Lists the variables accessed during the execution, including "Demo_Ada", "PKDATA.SCALE", "PKTASKING.CURRENT_DATA", "PKTASKING.TREGULATE.TMP", "PKTASKING.TSERVER.I", "PKTASKING.TSERVER.TMP", "PKUTIL.INJECTION", and "PKUTIL.NEW_ALTITUDE".

Labels with arrows point to the following components:

- Check details:** Points to the Check Details pane.
- Review statistics:** Points to the Review Statistics table.
- Run-time checks:** Points to the Results Explorer pane.
- Source code:** Points to the Source code editor.
- Variable access:** Points to the Variable Access pane.
- Call hierarchy:** Points to the Call Hierarchy pane.

You use the Results Manager perspective in the tutorial “Reviewing Verification Results” on page 4-2.

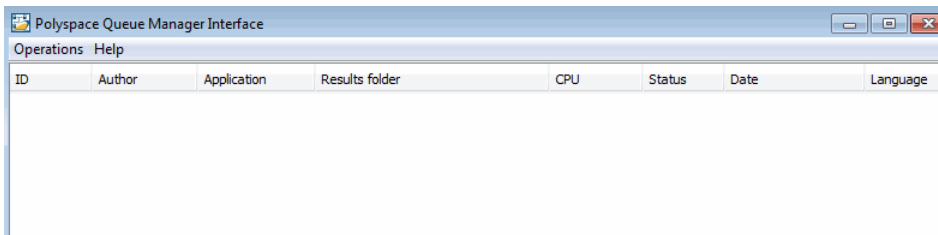
Other Polyspace Components

In addition to the Polyspace verification environment, Polyspace products provide other components to manage verifications, improve productivity, and track software quality. These components include:

- Polyspace Queue Manager Interface (Spooler)
- Polyspace Metrics Web Interface

Polyspace Queue Manager Interface (Polyspace Spooler)

The Polyspace Queue Manager (also called the Polyspace Spooler) is the graphical user interface of the Polyspace Server for Ada software. You use the Polyspace Queue Manager Interface to move jobs within the queue, remove jobs, monitor the progress of individual verifications, and download results.



You use the Polyspace Queue Manager in the tutorial “Starting Server Verification from Project Manager” on page 3-4.

Polyspace Metrics Web Interface

Polyspace Metrics is a web-based tool for software development managers, quality assurance engineers, and software developers. Polyspace Metrics allows you to evaluate software quality metrics, and monitor changes in code metrics and run-time checks through the lifecycle of a project.

For information on using Polyspace Metrics, see “About Polyspace Metrics”.

Working with Polyspace Software

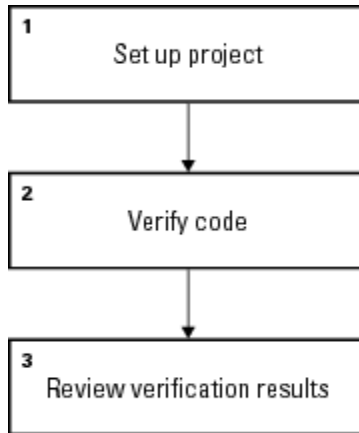
In this section...

“Basic Workflow” on page 1-11

“The Workflow in This Guide” on page 1-12

Basic Workflow

The basic workflow for using Polyspace software to verify Ada source code is:



In this workflow, you:

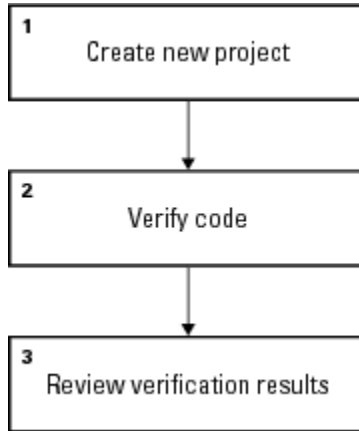
- 1** Use the Project Manager perspective to set up a project file.
- 2** Verify code on a server or client.

You can use the Project Manager perspective to start the verification or you can select files from a Microsoft® Windows® folder and send them to Polyspace software for verification. For verifications that run on a server, you can use the Polyspace Queue Manager Interface (Polyspace Spooler) to manage the verifications and download the results to a client.

- 3** Use the Results Manager perspective to review verification results.

The Workflow in This Guide

The tutorials in this guide take you through the basic workflow, including the different options for running verifications:



In this workflow, you will:

- 1** Create a new project that you can use for the other steps in the workflow. See the tutorial “Setting Up a Polyspace Project” on page 2-2.
- 2** Verify a single Ada source code package.

See the tutorial “Running a Verification” on page 3-2. In this tutorial, you will use the Project Manager to run a verification of the package on a server and a client.
- 3** Review the verification results. See the tutorial “Reviewing Verification Results” on page 4-2.

Learning More

In this section...
“Product Help” on page 1-13
“MathWorks Online” on page 1-13

Product Help

To access the help that came with your installation, select **Help > Help** or click the Help icon in the Polyspace window.

To access the online documentation for Polyspace products, go to:

www.mathworks.com/help/toolbox/polyspace/polyspace_product_page.html

MathWorks Online

For additional information and support, see:

www.mathworks.com/products/polyspace

Related Products

In this section...
“Polyspace Products for Verifying C/C++ Code” on page 1-14
“Polyspace Products for Linking to Models” on page 1-14

Polyspace Products for Verifying C/C++ Code

For information about Polyspace products that verify C/C++ code, see the following:

<http://www.mathworks.com/products/polyspaceclientc/>

<http://www.mathworks.com/products/polyspaceserverc/>

Polyspace Products for Linking to Models

For information about Polyspace products that link to models, see the following:

<http://www.mathworks.com/products/polyspacemodelsl/>

<http://www.mathworks.com/products/polyspaceumlrh/>

Setting Up Polyspace Project

Setting Up a Polyspace Project

In this section...
“About This Tutorial” on page 2-2
“Creating a New Project” on page 2-2

About This Tutorial

You must have a project file before you can run a Polyspace verification of your source code. In this tutorial, you will create a project that you can use to run verifications in later tutorials.

You will verify the package `example.adb` that comes with the Polyspace installation CD. You can learn more about the files and folders required for this tutorial in “Preparing Project Folders” on page 2-3.

Creating a New Project

- “What Is a Project?” on page 2-2
- “Preparing Project Folders” on page 2-3
- “Opening the Polyspace Verification Environment” on page 2-4
- “Creating a New Project to Verify an Ada Package” on page 2-6

What Is a Project?

In Polyspace software, a project is a named set of parameters for verification of your software project’s source files. A project includes:

- Source files
- Include folders
- Analysis options
- One or more Modules, each of which include:
 - Source (specific versions of source files used in the verification)
 - Configuration (specific set of analysis options used for the verification)

- Verification results

You can create your own project or use an existing one. You can create and modify a project using the Project Manager perspective.

In this tutorial, you create a new project and save it as a configuration file (.cfg).

Preparing Project Folders

Before you start verifying Ada code with Polyspace software, you must know the locations of the Ada source package and any other specifications upon which it may depend either directly or indirectly. You must also know where you want to store the verification results.

For each project, you decide where to store source files and results. For example, you can create a project folder and then create separate folders for the source files, include files, and results within the project folder.

For this tutorial, prepare a project folder as follows:

- 1** Create a project folder named `polyspace_project`.
- 2** Open `polyspace_project`, and create the following folders:
 - `sources`
 - `includes`
 - `results`

- 3** Copy the file `example.adb` and `example.ads` from

`Install_folder\Examples\Demo_Ada_Single-File\sources`

to

`polyspace_project\sources`

where `Install_folder` is the installation folder.

- 4** Copy all files from

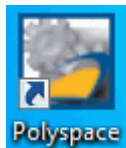
`Install_folder\Examples\Demo_Ada_Single-File\sources`
to
`polyspace_project\includes.`

Opening the Polyspace Verification Environment

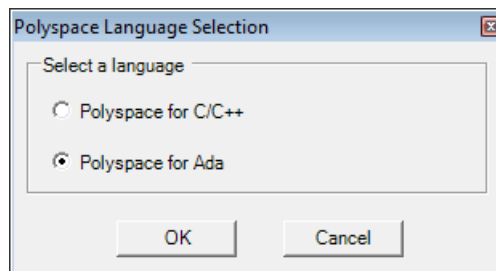
Use the Polyspace verification environment to create projects, start verifications, and review verification results.

To open the Polyspace verification environment:

- 1 Double-click the Polyspace icon.

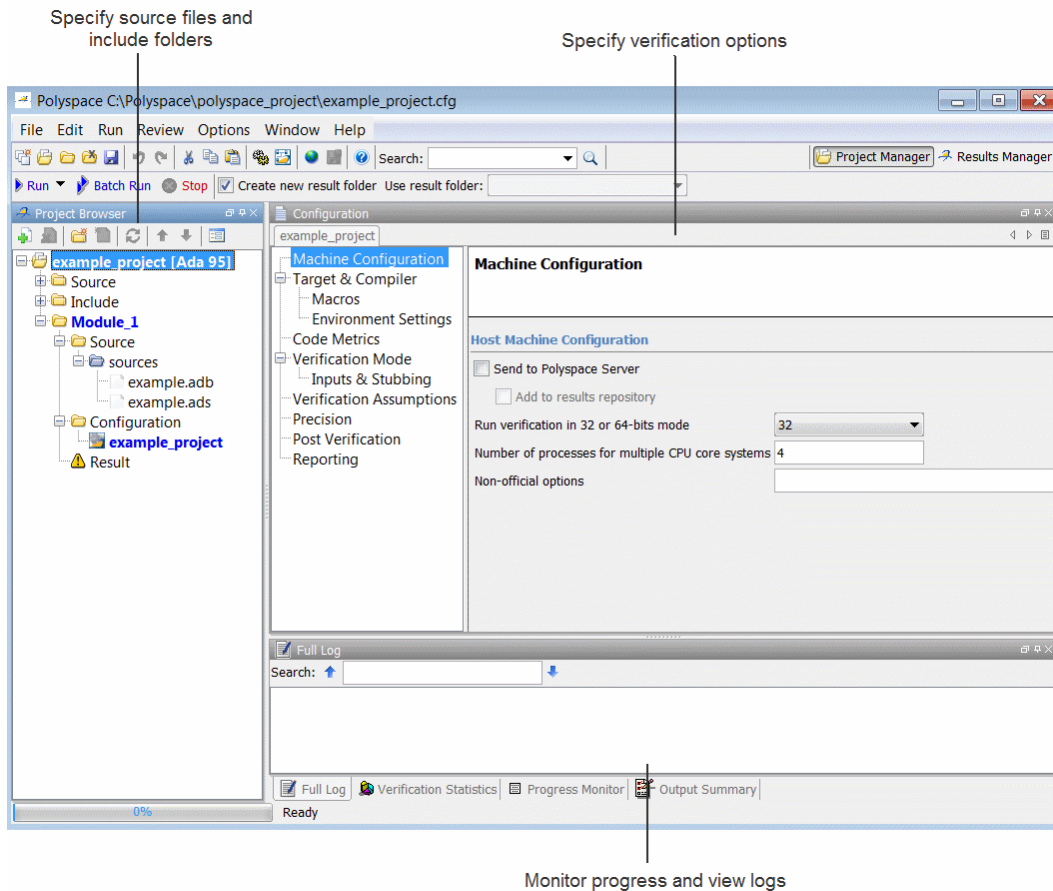


- 2 If you have both Polyspace Client for C/C++ and Polyspace Client for Ada products on your system, the Polyspace Language Selection dialog box opens.



- Select **Polyspace for Ada**, and click **OK**.

The Polyspace verification environment opens.



By default, the Polyspace verification environment displays the Project Manager perspective. The Project Manager perspective has three main sections.

Use this section...	For...
Project Browser (upper-left)	Specifying: <ul style="list-style-type: none">• Source files• Include folders• Results folder
Configuration (upper-right)	Specifying analysis options
Output (lower-right)	Monitoring the progress of a verification, and viewing status, log messages, and general verification statistics.

You can resize or hide any of these sections. You learn more about the Project Manager perspective later in this tutorial.

Creating a New Project to Verify an Ada Package

You must have a project, saved with file type `.cfg`, to run a verification. In this part of the tutorial, you create a new project to verify `example.adb`.

You create a new project by:

- “Opening a New project” on page 2-6
- “Specifying the Source Files and Include Folders” on page 2-8
- “Specifying the Analysis Options” on page 2-9
- “Specifying Source Files to Verify” on page 2-10
- “Saving the Project” on page 2-10

Opening a New project. To open a new project for verifying `example.adb`:

- 1 Select **File > New Project**. The Project - Properties dialog box opens.
- 2 In the **Project name** field, enter `example_project`.
- 3 Clear the **Default location** check box.

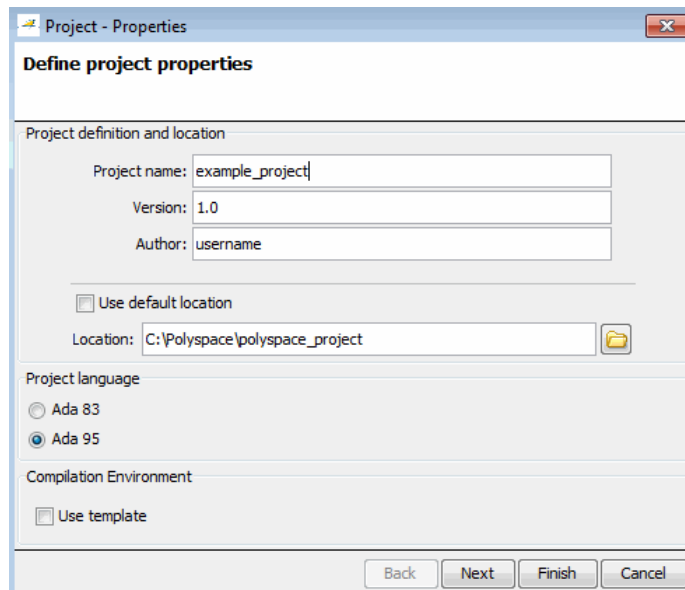
Note In this tutorial, you change the default location to the project folder that you created in “Preparing Project Folders” on page 2-3. Changing the default location makes it easier to specify source files and include folders.

You can update the default project location. Select **Options > Preferences**, which opens the Polyspace Preferences dialog box. On the **Project and result folder** tab, in the **Define default project location** field, specify the new default location.

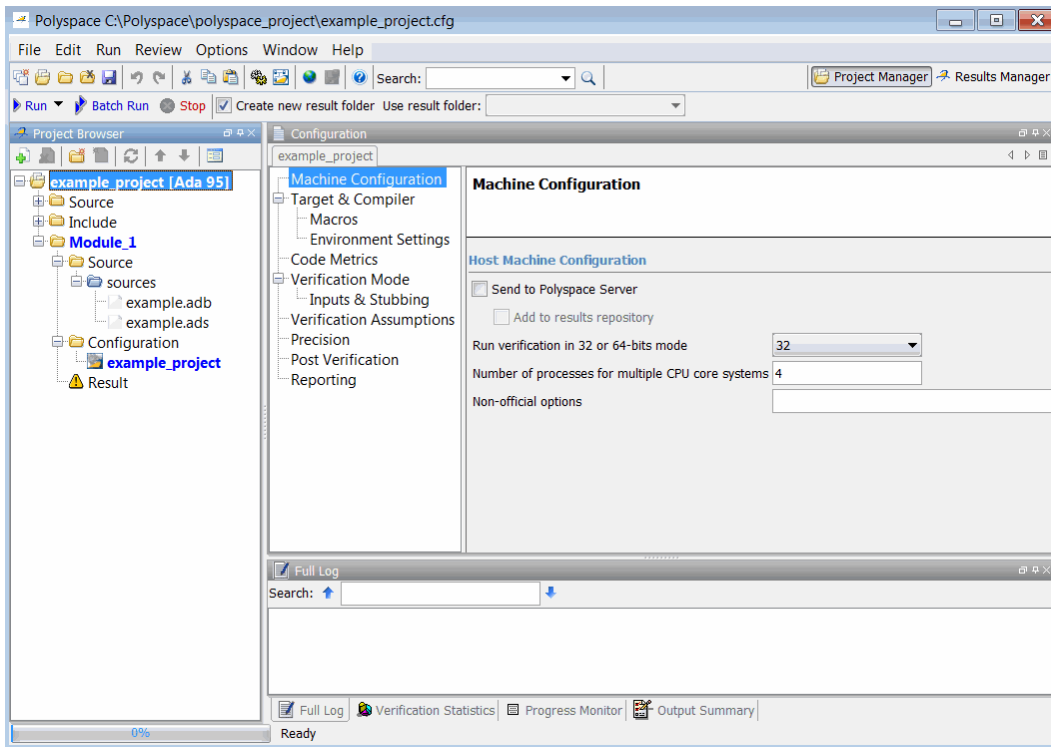
- 4 In the **Location** field, enter or navigate to the project folder that you created earlier.

In this example, the project folder is C:\Polyspace\polyspace_project.


- 5 Under **Project language**, select **Ada95**.



- 6 Click **Finish**. The example_project opens in the Polyspace verification environment.



Specifying the Source Files and Include Folders. To specify the source files and include folders for the verification of `example.adb`:

- 1 In the **Project Browser**, select the Source folder.
- 2 On the Project Browser toolbar, click the Add source icon . The Project - Add Source Files and Include Folders dialog box opens.
- 3 The project folder `polyspace_project` should appear in the **Look in** field. If it does not, navigate to that folder.
- 4 Select the folder `sources`. Then click **Add source**.

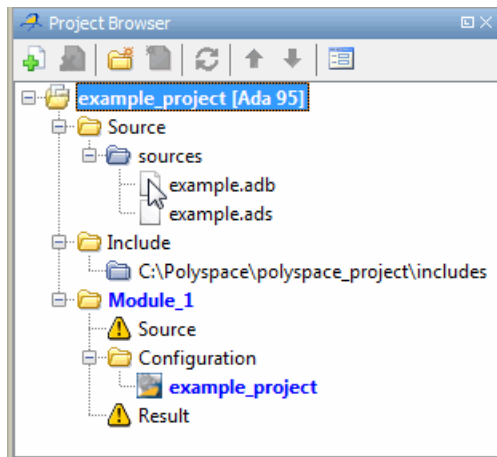
The files, `example.adb` and `example.ads`, appear in the Source tree for `example_project`.

5 Select the `includes` folder. Then click **Add Include**.

The `includes` folder appears in the Include tree for `example_project`.

6 Click **Finish** to apply the changes and close the dialog box.

The Project Browser now looks like the following graphic.



Specifying the Analysis Options. The analysis options in the **Configuration** pane of the Project Manager perspective include parameters that Polyspace software uses during the verification process. For more information about analysis options, see “Analysis Option Reference”.

To specify the analysis options for this tutorial:

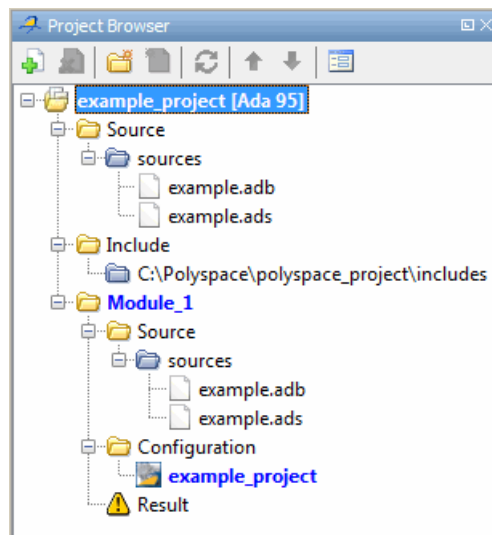
- 1** In the Project Manager perspective, select the **Configuration > Target & Compiler** pane.
- 2** From the **Target operating system** drop-down list, select `no-predefined-OS`.
- 3** Use the default values for all other options.

Specifying Source Files to Verify. Before you can start a verification, you must specify the files in the project that you want to verify. In `example_project`, there are two files to verify.

To specify source files for a verification:

- 1 In the Project Browser Source tree, right-click the folder `example_project[Ada 95] > Source > sources`, which contains the source files `example.adb` and `example.ads`.
- 2 From the context menu, select **Copy Source File to > Module_1**.

The source files `example.adb` and `example.ads` appear in the Source tree of `Module_1`.



Saving the Project. To save the project, select **File > Save (Ctrl+S)**.

Running Verification

Running a Verification

In this section...
“About This Tutorial” on page 3-2
“Preparing for Verification” on page 3-3
“Starting Server Verification from Project Manager” on page 3-4
“Starting Client Verification from Project Manager” on page 3-12

About This Tutorial

- “Overview” on page 3-2
- “Before You Start” on page 3-3

Overview

Once you have created the project `example_project.cfg` as described in “Creating a New Project” on page 2-2, you can run the verification.

You can run a verification on a server or a client.

Use...	For...
Server	<ul style="list-style-type: none"> • Best performance • Large files (more than 800 lines of code including comments)
Client	<ul style="list-style-type: none"> • When the server is busy • Small files <hr/> <p>Note Verification on a client takes more time. You might not be able to use your client computer when a verification is running on it.</p> <hr/>

In this tutorial, you learn how to run a verification on a server and a client, and you learn how to start a verification using the Project Manager.

The server and client verifications store the same results in your project. You review these results in the tutorial “Reviewing Verification Results” on page 4-2.

Before You Start

Before you start this tutorial, you must complete “Setting Up a Polyspace Project” on page 2-2. You use the folders and project file, `example_project.cfg`, from that tutorial to run the verifications.

Preparing for Verification

- “Opening the Project” on page 3-3
- “Specifying Source Files to Verify” on page 3-4

Opening the Project

To run a verification, you must have an open project file. For this tutorial, you use the project file `example_project.cfg` that you created in “Setting Up a Polyspace Project” on page 2-2. Open `example_project.cfg` if it is not already open.

To open `example_project.cfg`:

- 1** If the Polyspace verification environment is not already open, double-click the Polyspace icon.
- 2** Select **File > Open project**.

The Open a Polyspace project file dialog box opens.
- 3** In **Look in**, navigate to `polyspace_project`.
- 4** Select `example_project.cfg`.
- 5** Click **Open** to open the file and close the dialog box.

Specifying Source Files to Verify

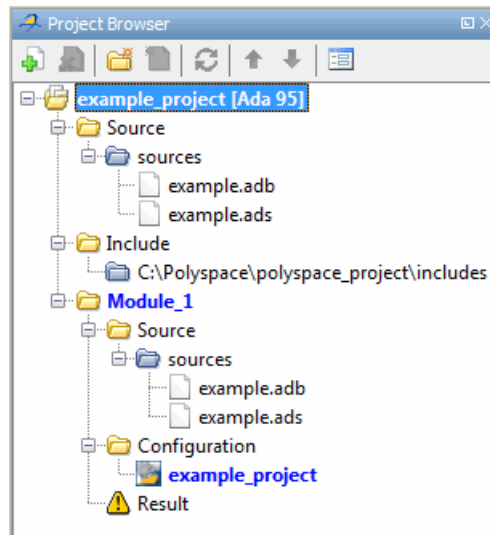
Each Polyspace project can contain multiple modules. Each of these modules verify a specific set of source files using a specific set of analysis options.

Therefore, before you can launch a verification, you must specify which files in your project you want to verify. In the `example_project` used in this tutorial, there is only one file to verify.

To copy source files to a module:

- 1 In the Project Browser Source tree, right-click the folder `example_project[Ada 95] > Source > sources`, which contains the source files `example.adb` and `example.ads`.
- 2 From the context menu, select **Copy Source File to > Module_1**.

The source files `example.adb` and `example.ads` appear in the Source tree of `Module_1`.



Starting Server Verification from Project Manager


- “Starting the Verification” on page 3-5

- “Monitoring the Progress of the Verification” on page 3-6
- “Removing Verification Results from the Server” on page 3-9
- “Troubleshooting a Failed Verification” on page 3-10

Starting the Verification

In this part of the tutorial, you run the verification on a server.

To start a verification that runs on a server:

- 1** In the Project Manager perspective, select the **Configuration > Machine Configuration** pane.
- 2** Select the **Send to Polyspace Server** check box.
- 3** On the Project Manager toolbar, click  .

Note If the verification fails, go to “Troubleshooting a Failed Verification” on page 3-10.

The verification has three main phases:

- a** Checking syntax and semantics (the compile phase). Because Polyspace software is independent of any particular Ada compiler, this phase shows whether your code is portable, maintainable, and complies with Ada standards.
- b** Generating a main if there is no main and the **Configuration > Verification Mode > Verify module** option is selected. For more information about generating a main, see “Verify module”.
- c** Analyzing the code for run-time errors and generating color-coded diagnostics.

The compile phase of the verification runs on the client. When the compile phase is complete:

- You see the message `queued on server` at the bottom of the Project Manager perspective. This message indicates that the part of the verification that takes place on the client is complete. The rest of the verification runs on the server.
- A message in **Output Summary** gives you the identification number (Analysis ID) for the verification.

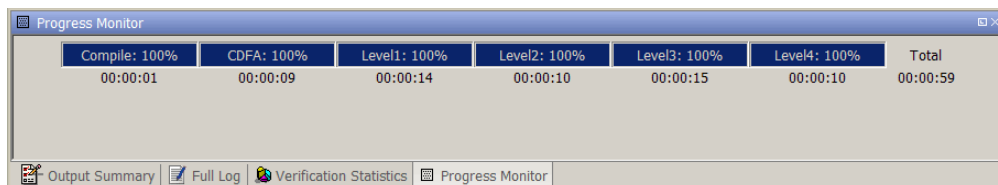
4 For information on any message in the log, click the message.

Monitoring the Progress of the Verification

There are two ways to monitor the progress of a verification:


- **Using the Project Manager** — Allows you to follow the progress of the verifications you submitted to the server, as well as client verifications.
- **Using the Queue Manager (Spooler)** — Allows you to follow the progress of any verification job in the server queue.

Monitoring Progress Using Project Manager. You can monitor the progress of your verification by viewing the progress monitor and logs at the bottom of the Project Manager perspective.




You can monitor the progress of the verification by watching the progress bar and viewing the logs at the bottom of the window. The progress monitor highlights the current phase in blue and displays the amount of time and completion percentage for that phase.

The logs report additional information about the progress of the verification. To view a log, click the button for that log. The information appears in the log display area at the bottom of the Project Manager window:

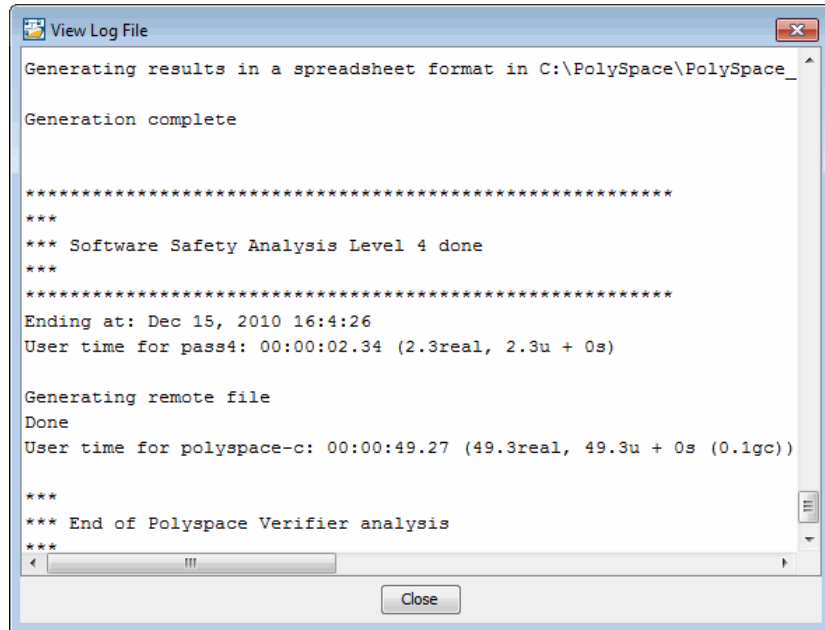
- Click the **Output Summary** tab to display compile phase messages and errors. You can search the log by entering search terms in the **Search** box and clicking the left arrow to search backward or the right arrow to search forward.
- Click the **Verification Statistics** tab to display statistics, such as analysis options, stubbed functions, and the verification checks performed.
- Click the **Refresh** button  to update the display as the verification progresses.
- Click the **Full Log** tab to display messages, errors, and statistics for all phases of the verification. You can search the log by entering search terms in the **Search** box and clicking the left arrow to search backward or the right arrow to search forward.

Monitoring Progress Using Queue Manager. You monitor the progress of the verification using the Polyspace Queue Manager (also called the Spooler).

To monitor the verification of `example_project.cfg`:

- 1** On the Polyspace verification environment toolbar, click the Polyspace Queue Manager icon . The Polyspace Queue Manager Interface opens.
- 2** Right-click the job you want to monitor. From the context menu, select **View log file**.

A window opens displaying the last one-hundred lines of the verification.




3 Click **Close** to close the window.

4 Select **Follow Progress** from the context menu. The Progress Monitor opens.

You can monitor the progress of the verification by watching the progress bar and viewing the logs at the bottom of the window. The progress monitor highlights the current phase in blue and displays the amount of time and completion percentage for that phase.

The logs report additional information about the progress of the verification. To view a log, click the button for that log. The information appears in the log display area at the bottom of the Project Manager window:

- Click the **Output Summary** tab to display compile phase messages and errors. You can search the log by entering search terms in the **Search** box and clicking the left arrow to search backward or the right arrow to search forward.

- Click the **Verification Statistics** tab to display statistics, such as analysis options, stubbed functions, and the verification checks performed.
- Click the **Refresh** button  to update the display as the verification progresses.
- Click the **Full Log** tab to display messages, errors, and statistics for all phases of the verification. You can search the log by entering search terms in the **Search** box and clicking the left arrow to search backward or the right arrow to search forward.

5 Select **File > Quit** to close the progress window.

6 Wait for the verification to finish.

When the verification is complete, the status in the Polyspace Queue Manager Interface changes from running to completed.

Removing Verification Results from the Server

At the end of a server verification, the server automatically downloads verification results to the results folder specified in the project. You do not need to manually download your results.

Note You can manually download verification results to another location on your client system, or to other client systems.

Verification results remain on the server until you remove them. Once your results have been downloaded to the client, you can remove them from the server queue.

To remove your results from the server:

- 1** In the Polyspace Queue Manager Interface, right-click the verification, and select **Remove From Queue**. A dialog box opens requiring confirmation that you want to remove the verification from the queue.
- 2** Click **Yes**.

Note To download the results and remove the verification from the queue, right-click the verification and select **Download Results And Remove From Queue**. If you download results before the verification is complete, you get partial results and the verification continues.

3 Select **Operations > Exit** to close the Polyspace Queue Manager Interface.

Once the results are on your client, you can review them using the Results Manager perspective. See “Reviewing Verification Results” on page 4-2.

Troubleshooting a Failed Verification

When you see a message that the verification failed, it indicates that Polyspace software could not perform the verification. The following sections present some possible reasons for a failed verification.

Hardware Does Not Meet Requirements. The verification fails if your computer does not have the minimal hardware requirements. For information about the hardware requirements, see

www.mathworks.com/products/polyspaceclientada/requirements.html.

To determine if this is the cause of the failed verification, search the log for the message:

```
Errors found when verifying host configuration.
```

You can:

- Upgrade your computer to meet the minimal requirements.
- Select the **Continue with current configuration option** in the General section of the Analysis options and run the verification again.

You Did Not Specify the Location of Included Files. If you see a message in the log, such as the following, either the files are missing or you did not specify the location of included files.

Verifier found an error in example.adb:23:14: "runtime_error (spec)" depends on "types (spec)"

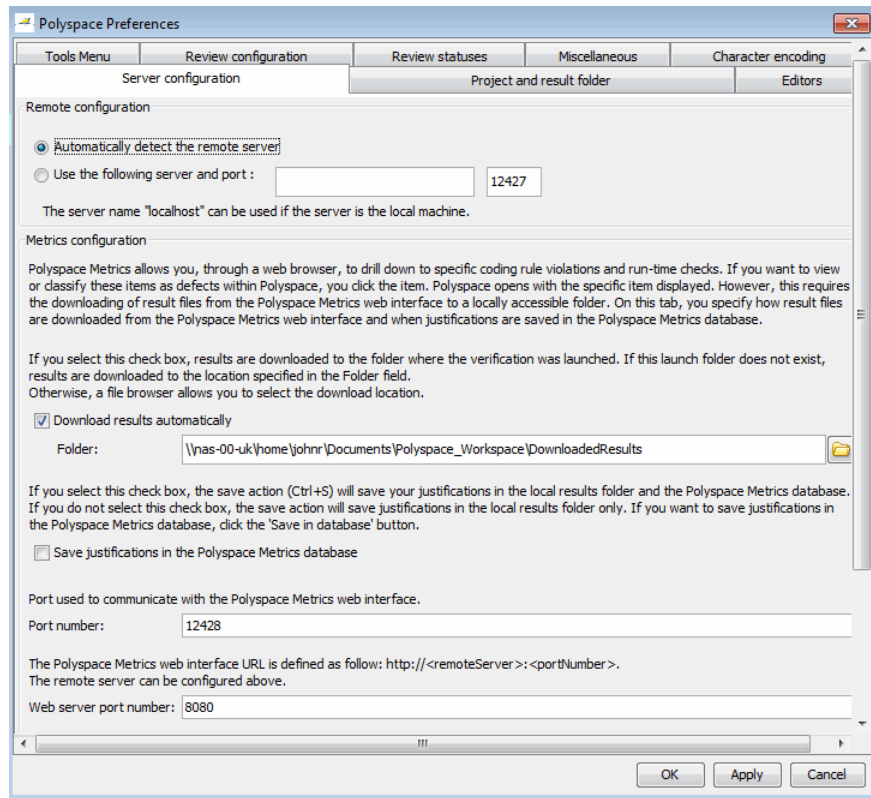
For information on how to specify the location of include files, see “Creating a New Project to Verify an Ada Package” on page 2-6.

Polyspace Software Cannot Find the Server. If you see the following message in the log, Polyspace software cannot find the server.

Error: Unknown host :

Polyspace software uses information in the preferences to locate the server. To find the server information in the preferences:

- 1** Select **Options > Preferences**.
- 2** Select the **Server Configuration** tab.



By default, Polyspace software automatically finds the server. You can specify the server by selecting **Use the following server and port** and providing the server name and port. For information about setting up a server, see “Software Installation”.

Starting Client Verification from Project Manager


- “Starting the Verification” on page 3-13
- “Monitoring the Progress of the Verification” on page 3-13
- “Completing Verification” on page 3-14
- “Stopping the Verification” on page 3-15

Starting the Verification

For the best performance, run verifications on a server. If the server is busy or you want to verify a small package, you can run a verification on a client.

Note Because a verification on a client can process only a limited number of variable assignments and function calls, the source code should have no more than 800 lines of code.

To start a verification that runs on a client:

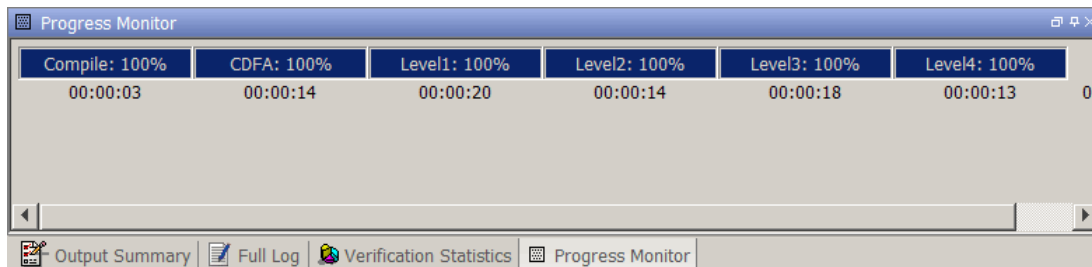
- 1 Open the project `example_project.cfg`. For information about opening a project, see “Preparing for Verification” on page 3-3.
- 2 In the Project Manager perspective, select the **Configuration > Machine Configuration** pane.
- 3 Clear the **Send to Polyspace Server** check box.
- 4 On the Project Manager toolbar, click . The Run button is a rectangular button with a blue play icon and the word 'Run' in blue text.
- 5 If you see a warning that Polyspace software will remove existing results from the results folder, click **Yes** to continue and close the message dialog box.

The **Output Summary** and **Progress Monitor** windows become active, allowing you to monitor the progress of the verification.

Note If the verification fails, go to “Troubleshooting a Failed Verification” on page 3-10.

Monitoring the Progress of the Verification

You can monitor the progress of the verification by viewing the progress monitor and logs at the bottom of the Project Manager perspective.



The progress bar highlights the current phase in blue and displays the amount of time and completion percentage for that phase.

The logs report additional information about the progress of the verification. To view a log, click the corresponding tab. The information appears in the log display area at the bottom of the Project Manager perspective. Follow the next steps to view the logs:

- 1 Click the **Output Summary** tab to display compile phase messages and errors. You can search the log by entering search terms in the **Search** field and clicking the left arrow to search backward or the right arrow to search forward.
- 2 Click the **Verification Statistics** tab to display statistics, such as analysis options, stubbed functions, and the verification checks performed.

Click the refresh button



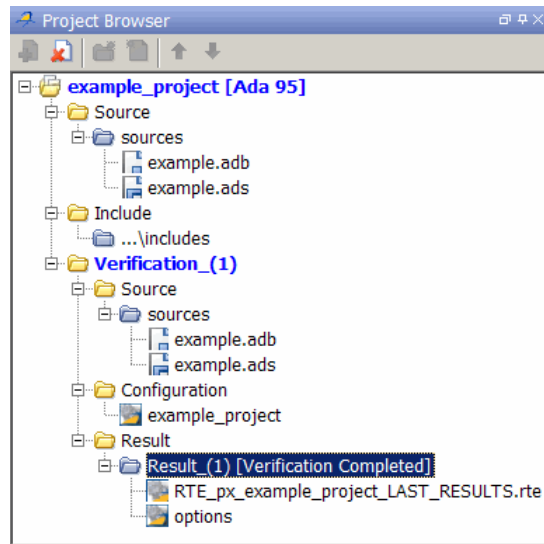
to update the display as the verification progresses.

- 3 Click the **Full Log** tab to display messages, errors, and statistics for all phases of the verification.

You can search the full log by entering a search term in the **Search in the log** box and clicking the left arrow to search backward or the right arrow to search forward.

Completing Verification

When the verification is complete, the results appear in the Project Browser.




In the tutorial “Reviewing Verification Results” on page 4-2, you open the Results Manager perspective and review the verification results.

Stopping the Verification

You can stop the verification before it is complete. If you stop the verification, results are incomplete. If you start another verification, the verification starts from the beginning.

To stop a verification:

- 1 On the Project Manager toolbar, click the **Stop** button  .

A warning dialog box opens, requesting confirmation.

- 2 Click **Yes**. The verification stops.
- 3 Click **OK** to close the **Message** dialog box.

Note Closing the Polyspace verification environment window does *not* stop the verification. To resume display of the verification progress, start the Polyspace software and open the project.

Reviewing Verification Results

Reviewing Verification Results

In this section...
“About This Tutorial” on page 4-2
“Opening Verification Results” on page 4-3
“Exploring Results Manager Perspective” on page 4-4
“Reviewing Results” on page 4-7
“Reviewing Results Systematically” on page 4-17
“Generating Reports of Verification Results” on page 4-22

About This Tutorial

- “Overview” on page 4-2
- “Prerequisites” on page 4-3

Overview

In the previous tutorial, “Running a Verification” on page 3-2, you completed a verification of the package `example.adb`. In this tutorial, you explore the verification results.

The Polyspace verification environment contains a Results Manager perspective that you use to review results. In this tutorial, you learn:

- How to use the Results Manager perspective, including how to:
 - Open the Results Manager perspective and view verification results.
 - Explore results.
 - Generate reports.
- How to interpret the color coding that Polyspace software uses to identify the severity of an error.
- How to find an error in the source code.

Prerequisites

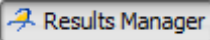
Before starting this tutorial, complete the tutorial “Running a Verification” on page 3-2. In this tutorial, you use the verification results stored in the following file:

```
polyspace_project\Module_1\Result_1\RTE_px_example_project_
LAST_RESULTS.rte
```

Opening Verification Results

- “Opening the Results Manager Perspective” on page 4-3
- “Opening Verification Results” on page 4-3

Opening the Results Manager Perspective

Use the Results Manager perspective to review verification results. To open the Results Manager perspective, on the Polyspace verification environment toolbar, click the **Results Manager** button .

Opening Verification Results

To open the verification results:

- 1** In the Polyspace verification environment, select **File > Open Result**.
The Open results dialog box opens.
- 2** Navigate to the results folder:

```
polyspace_project\Module_1\Result_1.
```
- 3** Select the file `RTE_px_example_project_LAST_RESULTS.rte`.
- 4** Click **Open**. The results appear in the Results Manager perspective.

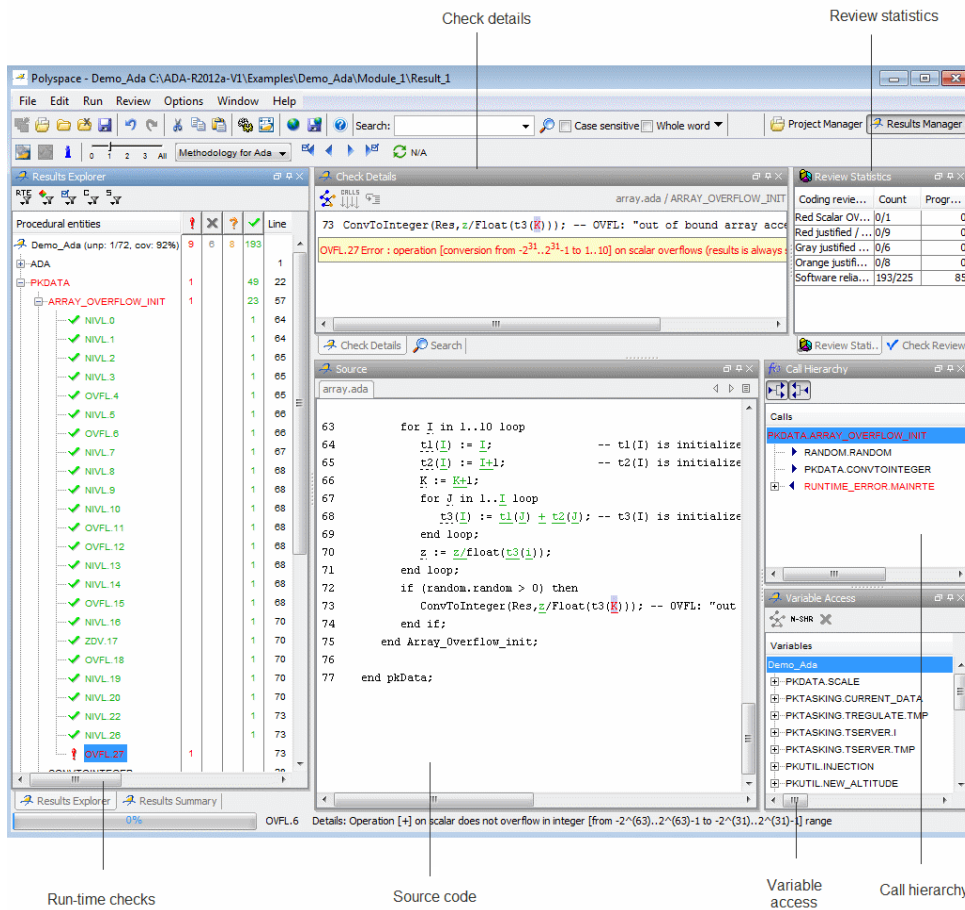
Note You can also open results from the Project Manager perspective by double-clicking the results file in the Project Browser.

Exploring Results Manager Perspective

- “Overview” on page 4-4
- “Reviewing Procedural Entities” on page 4-5

Overview

The Results Manager perspective looks like the following graphic.



The Results Manager perspective has six sections below the toolbar. Each section provides a different view of the results. The following table describes these views.

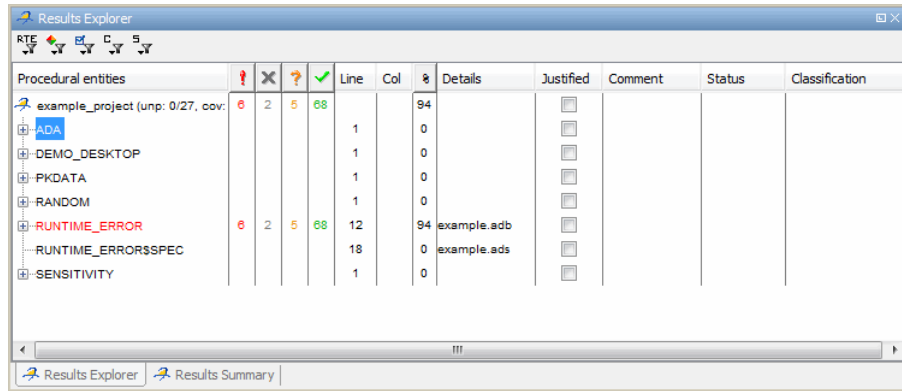
This Pane or View ...	Displays ...
Results Explorer\Results Summary (Procedural entities view)	Checks (diagnostics) for each file and procedure in the project
Source (Source code view)	Source code for a selected check in the procedural entities view
Check Details (Selected check view)	Details about the selected check
Check Review\Review Statistics (Coding review progress view)	Review information about the selected check Statistics about the review progress for checks with the same type and category as the selected check
Variable Access (Variables view)	Information about the global variables declared in the source code
Call Hierarchy (Call tree view)	Tree structure of function calls

You can resize or hide any of these sections.

Reviewing Procedural Entities

The **Results Explorer** view displays a table with information about the diagnostics for each file in the project. The **Results Explorer** view is also called the Procedural entities view.

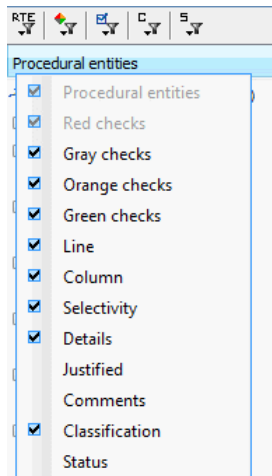
When you first open the results file from the verification of `example.adb`, you see the following procedural entities view.



The package `RUNTIME_ERROR` is red because it has run-time errors. Polyspace software assigns each package the color of the most severe error found in the package. The first column of the Procedural entities view contains the name of the procedural entity (package or function). The following table describes some of the other columns in the procedural entities view.

Column Heading	Indicates
	Number of red checks (operations where an error always occurs)
	Number of gray checks (unreachable code)
	Number of orange checks (warnings for operations where an error might occur)
	Number of green checks (operations where an error never occurs)
	Selectivity of the verification (percentage of checks that are not orange), which is an indication of the level of proof.

If you place the cursor over **Procedural entities** and right-click, you specify the columns that are displayed, for example, **Classification**.



What you select in the procedural entities view determines what is displayed in the other views. In the following examples, you learn how to use the views and see how they interact.

Reviewing Results

- “What are Review Levels?” on page 4-7
- “Reviewing All Checks” on page 4-9
- “Reviewing Example Checks” on page 4-12
- “Filtering Checks” on page 4-14

What are Review Levels?

To facilitate your review of verification results, Polyspace allows you to control the type and number of orange checks displayed in the **Procedural entities** and **Source** views of the Results Manager perspective. There are five levels at which you can review your results:

- 0 — The software displays red and gray checks. In addition, you can configure the software to display orange checks that are potential run-time errors. Through the **Polyspace Preferences > Review Configuration** tab, specify the categories of potential run-time errors that you want the

software to display. By default, the software does not display any orange checks at this level. See “Reviewing Checks at Level 0” on page 4-18.

This level is suitable for the review of fresh code.

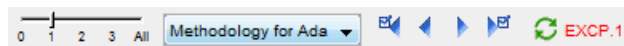
- 1, 2, and 3 — The software displays red, gray, and green checks. In addition, the software displays orange checks according to values specified on the **Polyspace Preferences > Review Configuration**. You can use either a predefined or a custom methodology to specify the number of orange checks per check category. See “Reviewing Checks at Levels 1, 2, and 3” on page 4-19.

For a predefined methodology, these levels are suitable for reviews at the following stages of the development process.

Level	Development Stage
1	Fresh code
2	Unit tested code
3	Code Review

- All — In addition to red, gray, and green checks, the software displays *all* orange checks. Use this level when you want to carry out an exhaustive review of your verification results. See “Reviewing All Checks” on page 4-9.

In the Results Manager perspective, the toolbar provides controls specific to review levels.



The controls include:

- A slider for selecting the review level — 0, 1, 2, 3, or All. By default, the Results Manager perspective opens at level 1.
- A menu for selecting the review methodology for levels 1, 2, and 3.
- Arrows for navigating through checks.

Reviewing All Checks

In this part of the tutorial, you learn how to use the Results Manager perspective views to examine checks from a verification:

- “Selecting a Check to Review” on page 4-9
- “Displaying the Calling Sequence” on page 4-11
- “Tracking Review Progress” on page 4-11

By default, the Results Manager perspective opens at level 1. To display all checks in the Procedural entities view, move the Review Level slider to All.

Selecting a Check to Review. In the procedural entities view, `RUNTIME_ERROR` is red, indicating that this package has at least one red check. To review a red check in `RUNTIME_ERROR`:

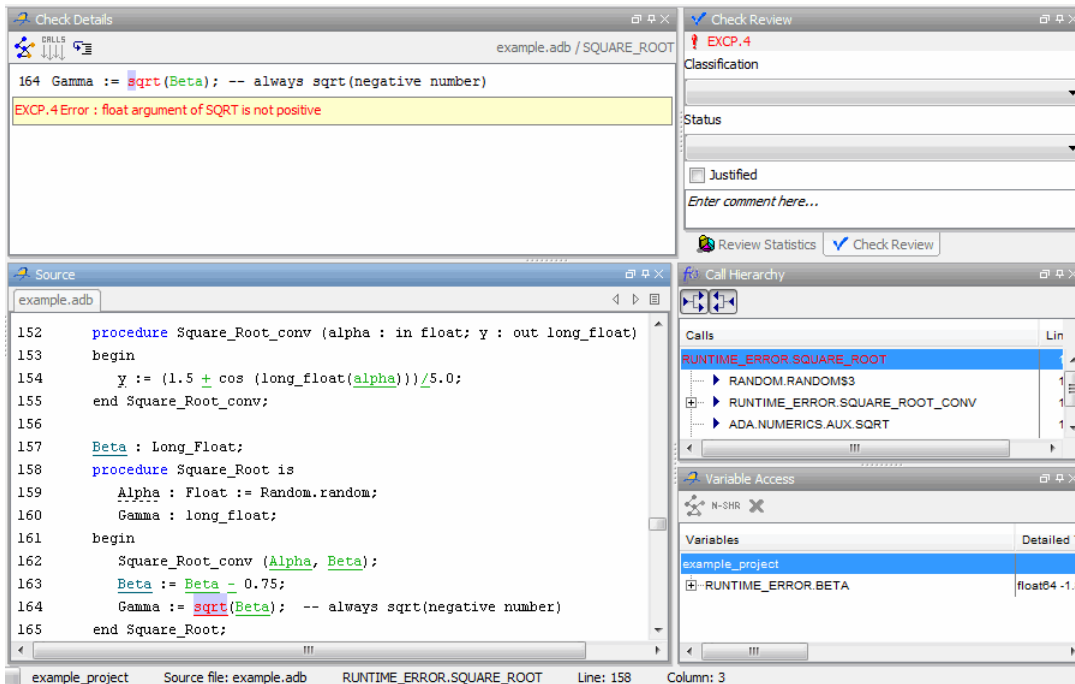
- 1 In the procedural entities view, expand `RUNTIME_ERROR`.
- 2 Expand the red function `SQUARE_ROOT`.



In the list of checks, each item has an acronym and a number. The acronym identifies the check type. For example, in `EXCP.4`, `EXCP` stands for Arithmetic Exception. For more information about different types of checks, see “Run-Time Check Reference”.

- 3 Click the red check, `EXCP.4`.

4 Reviewing Verification Results



You see the section of source code where this error occurs, and details about the check in the **Check Details** pane..

4 On the **Check Review** tab, you can:

- Classify the run-time check as a defect. From the **Classification** drop-down list, select a category, for example, High.
- Assign a status, for example, Fix. This action indicates to Polyspace that you have reviewed the check.
- Justify the check. For example, if you classified the check as Not a defect, you could select the **Justified** check box to indicate that the check is justified.
- In the **Comment** field, enter remarks, for example, defect or justification information.

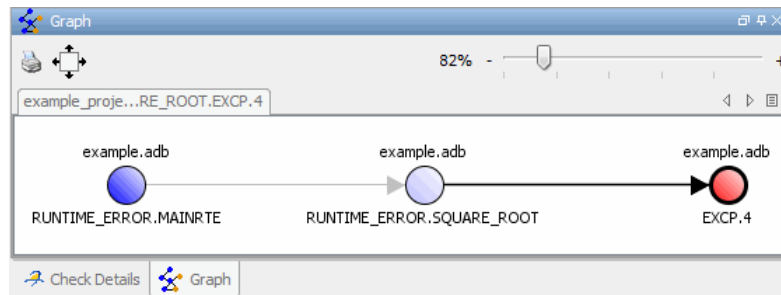
Displaying the Calling Sequence. You can display the calling sequence that leads to the code associated with a check. To see the calling sequence for the red EXCP.4 check in SQUARE_ROOT:

- 1 In the procedural entities view, expand SQUARE_ROOT.
- 2 Click the red check EXCP.4.

- 3 In the **Check Details** toolbar, click the call graph button.



A window displays the call graph.



The code associated with EXCP.4 is in SQUARE_ROOT. The function MAINRTE calls SQUARE_ROOT.

Tracking Review Progress. Review Statistics allows you to keep track of the checks that you have reviewed.

Coding review progress	Count	Progress
Red EXCP justified / to justify	0/1	0
Red justified / to justify	0/6	0
Gray justified / to justify	0/2	0
Orange justified / to justify	0/5	0
Software reliability indicator	68/83	81

The **Count** column displays a ratio. The **Progress** column displays the equivalent percentage. The first row displays the ratio of justified checks to the total number of checks that have the same color and category as the current check. In this example, the first row displays the ratio of justified red EXCP checks to total red EXCP errors in the project.

The second, third, and fourth rows displays the ratio of justified checks to total checks for red, gray, and orange checks respectively. The fifth row displays the ratio of the number of green checks to the total number of checks, providing an indicator of the reliability of the software.

Reviewing Example Checks

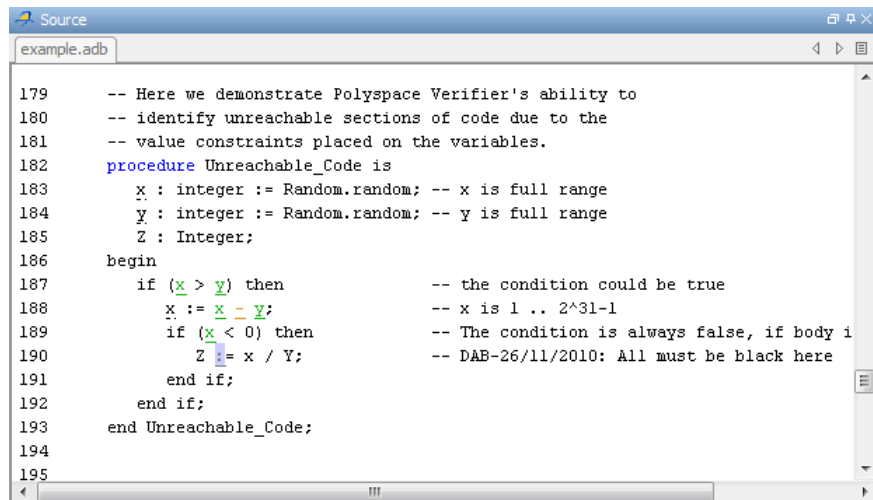
In this part of the tutorial, you learn about other types and categories of errors by reviewing the following checks in `example.adb`:

- “Example: Unreachable Code” on page 4-12
- “Example: A Function with No Errors” on page 4-13
- “Example: Division by Zero” on page 4-13

Example: Unreachable Code. Unreachable code is code that never executes. Polyspace software displays unreachable code in gray. In the following steps, you see an example of unreachable code.

- 1 Under **Procedural Entities**, expand `UNREACHABLE_CODE` and click the gray `UNR.10`.

You see the source code for this function in the source code view.



```
Source
example.adb
179  -- Here we demonstrate Polyspace Verifier's ability to
180  -- identify unreachable sections of code due to the
181  -- value constraints placed on the variables.
182  procedure Unreachable_Code is
183     x : integer := Random.random; -- x is full range
184     y : integer := Random.random; -- y is full range
185     z : Integer;
186  begin
187     if (x > y) then           -- the condition could be true
188         x := x - y;          -- x is 1 .. 2^31-1
189         if (x < 0) then      -- The condition is always false, if body i
190             z := x / y;     -- DAB-26/11/2010: All must be black here
191         end if;
192     end if;
193  end Unreachable_Code;
194
195
```

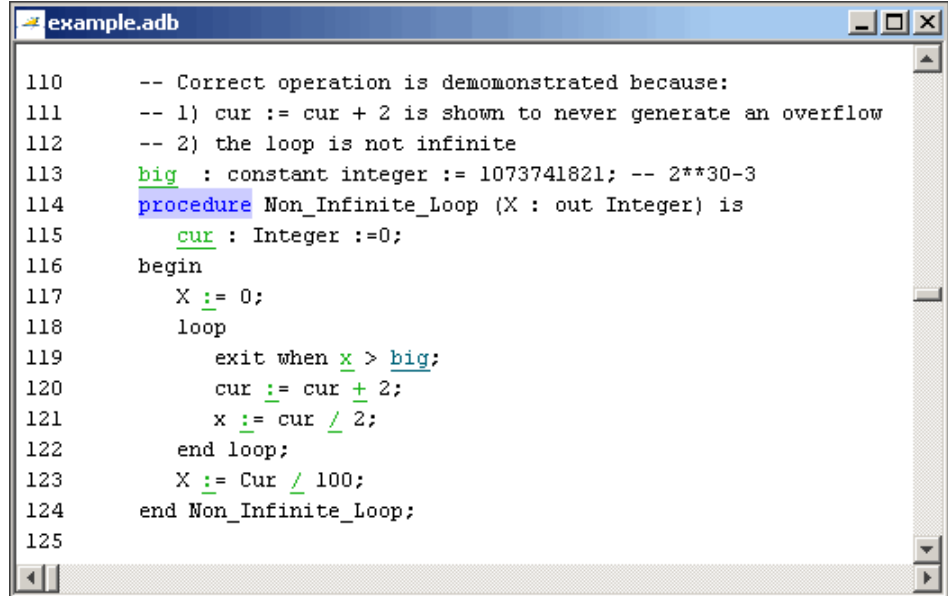
2 Examine the source code.

At line 190, the code `Z := x / Y;` is never reached because the condition `x < 0` is always false.

Example: A Function with No Errors. In the following example, Polyspace software determines, in code with a large number of iterations, that a loop terminates and a variable does not overflow:

1 In **Procedural entities**, click the green `NON_INFINITE_LOOP` function.

The source code for this function is displayed in the source code view.



```

example.adb
110  -- Correct operation is demononstrated because:
111  -- 1) cur := cur + 2 is shown to never generate an overflow
112  -- 2) the loop is not infinite
113  big : constant integer := 1073741821; -- 2**30-3
114  procedure Non_Infinite_Loop (X : out Integer) is
115      cur : Integer :=0;
116  begin
117      X := 0;
118      loop
119          exit when x > big;
120          cur := cur + 2;
121          x := cur / 2;
122      end loop;
123      X := Cur / 100;
124  end Non_Infinite_Loop;
125

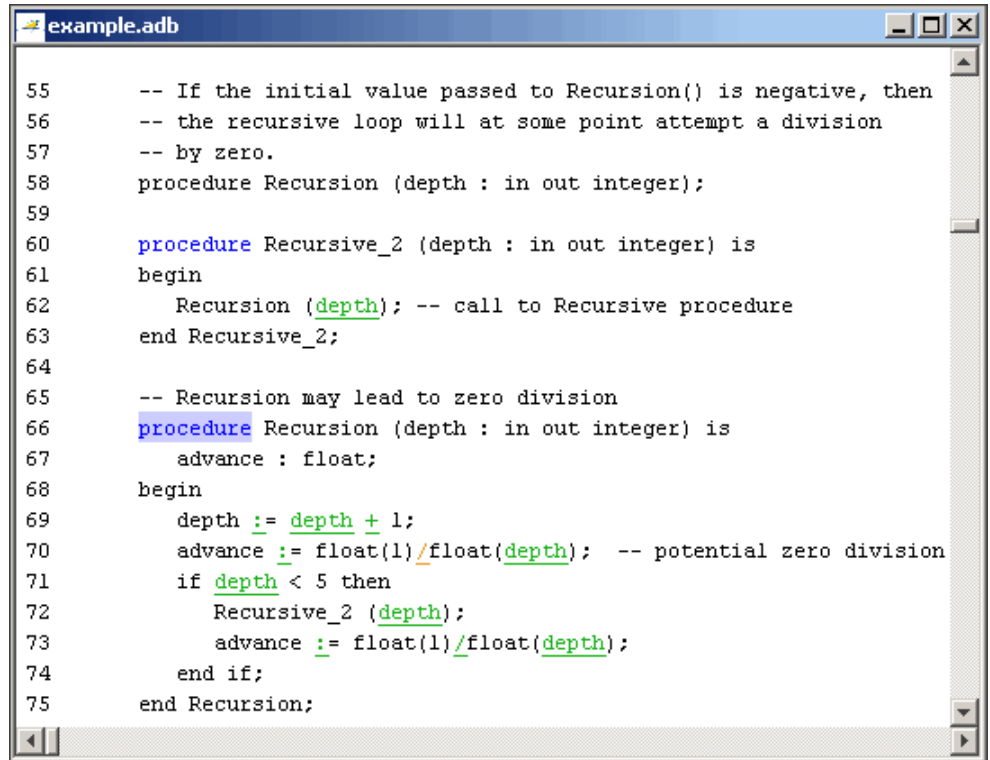
```

2 Examine the source code. The variable `cur` never overflows because the loop at line 117 terminates before `cur` can overflow.

Example: Division by Zero. In the following example, Polyspace software detects a potential division by zero:

1 In **Procedural entities**, expand `RECURSION`.

The source code for this function is displayed in the source code view.



```
55      -- If the initial value passed to Recursion() is negative, then
56      -- the recursive loop will at some point attempt a division
57      -- by zero.
58      procedure Recursion (depth : in out integer);
59
60      procedure Recursive_2 (depth : in out integer) is
61      begin
62          Recursion (depth); -- call to Recursive procedure
63      end Recursive_2;
64
65      -- Recursion may lead to zero division
66      procedure Recursion (depth : in out integer) is
67      advance : float;
68      begin
69          depth := depth + 1;
70          advance := float(1)/float(depth); -- potential zero division
71          if depth < 5 then
72              Recursive_2 (depth);
73              advance := float(1)/float(depth);
74          end if;
75      end Recursion;
```

2 Examine the RECURSION function.

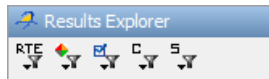
When RECURSION is called with depth less than zero, the code at line 70 results in division by zero. The orange color indicates that this result is a potential error (depending on the value of depth).

Filtering Checks

You can filter the checks that you see in the Results Manager perspective to focus on certain checks. Polyspace software allows you to filter your results in several ways. You can filter by:

- Check category (ZDV, IDP, NIP, etc.)
- Color of check (gray, orange, green)
- Justified or unjustified
- Classification
- Status

To filter checks, on the **Results Explorer** or **Results Summary** toolbar, select one of the filter buttons.



Tip The tooltip for a filter button describes what filter the button activates.

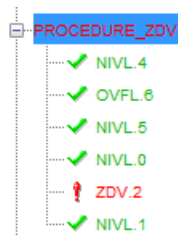
Example: Filtering NIVL Checks

You can use an RTE filter to hide a given check category, such as NIVL. When a filter is enabled, you do not see that check category.

To filter NIVL checks:

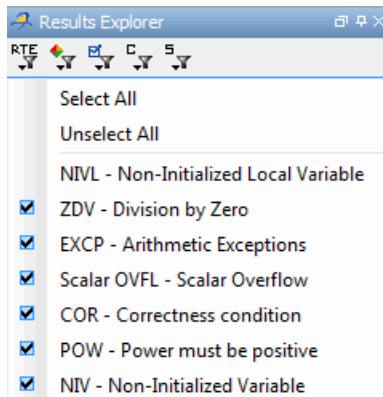
- 1 Expand PROCEDURE_ZDV.

PROCEDURE_ZDV has six checks; five are green and one is red.



- 2 Click the **RTE filter** icon .

3 Clear the NIVL check box.



The software hides the NIVL checks for PROCEDURE_ZDV.



4 Select the NIVL option to redisplay the NIVL check.

Note When you filter a check category, red checks of that category are not hidden. For example, if you filter ZDV checks, you still see ZDV.2 under PROCEDURE_ZDV.

Example: Filtering Green Checks

You can use a color filter to hide checks of a certain color.

To filter green checks:

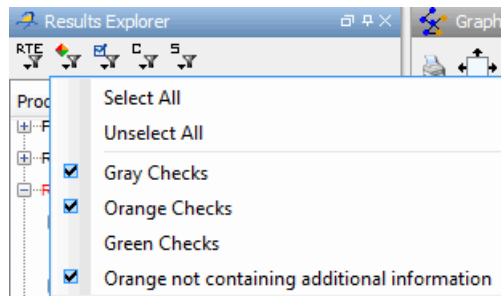
1 Expand PROCEDURE_ZDV.

PROCEDURE_ZDV has six checks; five are green and one is red.



2 Click the **Color filter** icon .

3 Clear **Green Checks**.



The software hides the green checks.



Reviewing Results Systematically

- “Reviewing Checks at Level 0” on page 4-18
- “Reviewing Checks at Levels 1, 2, and 3” on page 4-19
- “Reviewing Checks Progressively” on page 4-21

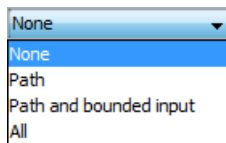
Reviewing Checks at Level 0

In addition to red and gray checks, at level 0, you can focus on orange checks that Polyspace identifies as potential run-time errors. These potential run-time errors fall into three categories:

- **Path** — The software identifies orange checks that are path-related issues, which are not dependent on input values.
- **Path and bounded input** — Default. In addition to orange checks that are path-related issues, the software identifies orange checks that are related to input values bounded by data range specifications (DRS).
- **All** — In addition to path-related and bounded input orange checks, the software identifies orange checks that are related to unbounded input values.

To specify the potential run-time error category for level 0:

- 1** In the Polyspace verification environment, select **Options > Preferences**. The Polyspace Preferences dialog box opens.
- 2** Select the **Review configuration** tab.
- 3** From the **Level** drop-down list, select your category.



The default is **Path and bounded input**. If you select **None**, the software displays only red and gray checks.

- 4** Click **OK** to save your options and close the Polyspace Preferences dialog box.

To select review level 0, on the Run-Time Checks toolbar, move the Review Level slider to **0**.

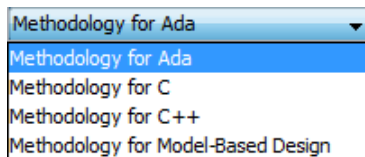
Reviewing Checks at Levels 1, 2, and 3

In addition to red, gray, and green checks, the software displays orange checks according to values specified on the **Review Configuration** tab in the Polyspace Preferences dialog box. See “Viewing Methodology Requirements for Levels 1, 2, and 3” on page 4-19

You can use either a predefined or a custom methodology to specify the number of orange checks per check category.

To select a predefined methodology and review level:

- 1 From the Results Manager perspective, select **Options > Preferences**.
- 2 In the Polyspace Preferences dialog box, select the **Review configuration** tab.
- 3 From the **Methodology** drop-down list, select, for example, Methodology for Ada.



- 4 Move the Review Level slider to your required level, for example, level 1.



Viewing Methodology Requirements for Levels 1, 2, and 3. In this part of the tutorial, you examine **Methodology for Ada**, which defines the number of orange checks that you review at levels 1, 2, or 3.

To examine the configuration for **Methodology for Ada**:

- 1 In the Polyspace verification environment, select **Options > Preferences**.
- 2 In the Polyspace Preferences dialog box, select the **Review configuration** tab.

3 From the **Methodology** drop-down list, select **Methodology** for Ada.


In the section **Levels 1, 2, and 3**, a table shows the number of orange checks that you review for a given level and check category.

Levels 1, 2, and 3			
	Level 1	Level 2	Level 3
Common			
ZDV	10	20	ALL
NIVL	AUTO	50	ALL
Scalar OVFL	AUTO	50	ALL
COR	AUTO	10	10
NIV	AUTO	5	10
Float OVFL	5	10	20
ASRT	AUTO	5	20
C & C++ only			
OBAI			
SHF			
IDP			
NIP			
STD_LIB			
ABS_ADDR			
C only			
IRV			
C++ only			
NNT			
CPP			
FRV			
OOP			
EXC			
Ada only			
EXCP			5
POW	AUTO	10	ALL

For example, when you select level 1, the table specifies that you review ten orange ZDV checks. The number of checks increases as you move from level 1 to level 3, reflecting the changing review requirements as you move through the development process.

4 Click **OK** to close the dialog box.

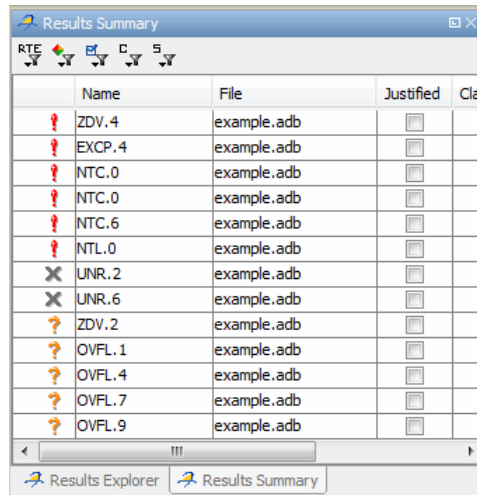
Reviewing Checks Progressively

On the Results Manager toolbar, use the forward arrow  to move to the next unjustified check. The software takes you through checks in the following order:


- 1 All red checks
- 2 All blocks of gray checks (the first check in each unreachable function)
- 3 Orange checks according to the selected methodology and criterion level

Earlier in this tutorial, you selected Methodology for Ada, criterion 1. In this part of the tutorial, you continue to review the checks for `example.adb` using this methodology and criterion level. To navigate through these checks:

- 1 Select the **Results Summary** view.



	Name	File	Justified	Clas
!	ZDV.4	example.adb	<input type="checkbox"/>	
!	EXCP.4	example.adb	<input type="checkbox"/>	
!	NTC.0	example.adb	<input type="checkbox"/>	
!	NTC.0	example.adb	<input type="checkbox"/>	
!	NTC.6	example.adb	<input type="checkbox"/>	
!	NTL.0	example.adb	<input type="checkbox"/>	
×	UNR.2	example.adb	<input type="checkbox"/>	
×	UNR.6	example.adb	<input type="checkbox"/>	
?	ZDV.2	example.adb	<input type="checkbox"/>	
?	OVFL.1	example.adb	<input type="checkbox"/>	
?	OVFL.4	example.adb	<input type="checkbox"/>	
?	OVFL.7	example.adb	<input type="checkbox"/>	
?	OVFL.9	example.adb	<input type="checkbox"/>	

- 2 Click the forward arrow .

The software displays ZDV.4 as the current check.

The source code view displays the source for this check. In the **Check Details** pane, you see information about this check.

Note You can display the calling sequence and track review progress as you did in “Reviewing Results” on page 4-7.

3 Continue to click the forward arrow until you have reviewed all the checks.

After the last check, a dialog box opens asking if you want to start again from the first check.

4 Click **No**.

Generating Reports of Verification Results

- “Polyspace Report Generator Overview” on page 4-22
- “Generating Report for example.adb” on page 4-23

Polyspace Report Generator Overview

The Polyspace Report Generator allows you to generate reports about your verification results, using predefined report templates.

The Polyspace Report Generator provides the following report templates:

- **Coding Rules Report** — Provides information about compliance with MISRA-C Coding Rules, as well as Polyspace configuration settings for the verification.
- **Developer Report** — Provides information useful to developers, including summary results, detailed lists of red, orange, and gray checks, and Polyspace configuration settings for the verification.

Developer Review Report – Provides the same information as the Developer Report, but reviewed results are sorted by review classification (High, Medium, Low, Not a defect) and status, and untagged checks are sorted by file location.

- **Developer with Green Checks Report** — Provides the same content as the Developer Report, but also includes a detailed list of green checks.
- **Quality Report** — Provides information useful to quality engineers, including summary results, statistics about the code, graphs showing distributions of checks per file, and Polyspace configuration settings for the verification.
- **Software Quality Objectives Report** — Provides information on software quality objectives (SQO), including code metrics, code verification (run-time checks), and configuration settings for the verification. The code metrics section provides the same information as the Code Metrics view of the Polyspace Metrics web interface.

The Polyspace Report Generator allows you to generate verification reports in the following formats:

- HTML
- PDF
- RTF
- DOC (Microsoft Word version 2003 or later)
- XML

Note With UNIX platforms, the Microsoft Word format is not supported. Use the RTF format instead.

Generating Report for example.adb

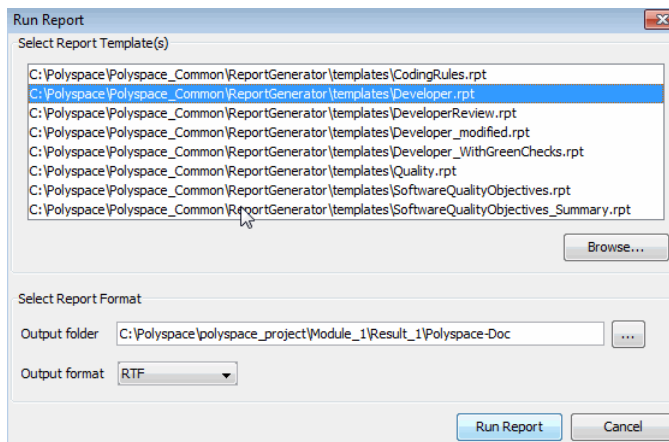
You can generate reports for any verification results using the Polyspace Report Generator.

To generate a verification report:

- 1 Open your verification results.

2 Select **Run > Run Report > Run Report**.

The Run Report dialog box opens.



3 In the Select Report Template section, select **Developer.rpt**.

4 In the Output folder section, select the polyspace_project folder.

5 From the **Output format** drop-down list, select, for example, RTF .

6 Click **Run Report**.

The software creates the specified report. When report generation is complete, the report opens.

A

analysis options 2-9
ANSI compliance 3-5

C

call graph 4-11
call tree view 4-4
calling sequence 4-11
cfg. *See* configuration file
client 1-7 3-2
 verification on 3-12
coding review progress view 4-4 4-11
Coding Rules perspective 1-7
color-coding of verification results 1-4 to 1-5 4-5
compile log
 Project Manager 3-6 3-13
 Spooler 3-7
compile phase 3-5
compliance
 ANSI 3-5
configuration file
 definition 2-2

D

desktop file
 definition 2-2
division by zero
 example 4-13
downloading
 results 3-9
dsk. *See* desktop file

F

files
 includes 2-8
 source 2-8
folders

includes 2-8
sources 2-8

H

hardware requirements 3-10
help
 accessing 1-13

L

logs
 compile
 Project Manager 3-6 3-13
 Spooler 3-7
 full
 Project Manager 3-6 3-13
 Spooler 3-7
 stats
 Project Manager 3-6 3-13
 Spooler 3-7
 viewing
 Project Manager 3-6 3-13
 Spooler 3-7

P

Polyspace products for Ada
 components 1-7
 overview 1-4
 related products 1-14
 user interface 1-7
 workflow 1-11
Polyspace Queue Manager Interface. *See* Spooler
Polyspace verification environment
 opening 2-4
preferences
 Project Manager perspective
 default server mode 3-5
 server detection 3-11
procedural entities view 4-4

- product overview 1-4
- progress bar
 - Project Manager window 3-6 3-13
- project
 - creation 2-2 2-6
 - definition 2-2
 - file types
 - configuration file 2-2
 - desktop file 2-2
 - folders
 - includes 2-3
 - results 2-3
 - sources 2-3
 - opening 3-3
 - saving 2-10
- Project Manager
 - monitoring verification progress 3-6 3-13
 - starting verification on client 3-12
 - starting verification on server 3-4
 - viewing logs 3-6 3-13
 - window 2-4
 - overview 2-4
 - progress bar 3-6 3-13
- Project Manager perspective 1-7
 - stopping 3-14

R

- related products 1-14
 - Polyspace products for linking to Models 1-14
 - Polyspace products for verifying C code 1-14
 - Polyspace products for verifying C++ code 1-14
- reports
 - generation 4-22
- results
 - downloading from server 3-9
 - opening 4-3
 - report generation 4-22
 - reviewing 4-1

- Results Manager perspective
 - call tree view 4-4
 - coding review progress view 4-4
 - opening 4-3
 - procedural entities view 4-4
 - selected check view 4-4
 - source code view 4-4
 - variables view 4-4
 - window
 - overview 4-4
- rte view. *See* procedural entities view
- Run-Time Checks perspective 1-7

S

- selected check view 4-4
- server 1-7 3-2
 - detection 3-11
 - information in preferences 3-11
 - installation 3-11
 - verification on 3-4
- source code view 4-4
- Spooler 1-7
 - monitoring verification progress 3-7
 - removing verification from queue 3-9
 - use 3-7
 - viewing log 3-7

T

- troubleshooting failed verification 3-10

U

- unreachable code
 - example 4-12

V

- variables view 4-4
- verification

- Ada code 1-4
- C code 1-14
- C++ code 1-14
- client 3-2
- compile phase 3-5
- failed 3-10
- monitoring progress
 - Project Manager 3-6 3-13
 - Spooler 3-7
- phases 3-5
- results
 - color-coding 1-4 to 1-5
 - opening 4-3
 - report generation 4-22
 - reviewing 4-1

- running 3-2
- running on client 3-12
- running on server 3-4
- starting
 - from Project Manager 3-2 3-13
 - from Project Manager perspective 3-5
- stopping 3-15
- troubleshooting 3-10

W

- workflow
 - basic 1-11
 - in this guide 1-12