# Polyspace® Products for C 8 Getting Started Guide



#### **How to Contact MathWorks**



(a)

www.mathworks.comWebcomp.soft-sys.matlabNewsgroupwww.mathworks.com/contact\_TS.htmlTechnical Support

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



508-647-7001 (Fax)

508-647-7000 (Phone)

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

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

Polyspace<sup>®</sup> Products for C Getting Started Guide

© COPYRIGHT 1997–2011 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 6.0 (Release 2008b) |
| March 2009     | Third printing  | Revised for Version 7.0 (Release 2009a) |
| September 2009 | Online only     | Revised for Version 7.1 (Release 2009b) |
| March 2010     | Online only     | Revised for Version 7.2 (Release 2010a) |
| September 2010 | Fourth Printing | Revised for Version 8.0 (Release 2010b) |
| April 2011     | Fifth Printing  | Revised for Version 8.1 (Release 2011a) |
|                |                 |   |

# Contents

## Introduction to Polyspace Products for Verifying C Code

| Product Overview   | 1-2  |
|--|------|
| Overview of Polyspace Verification                                 | 1-2  |
| The Value of Polyspace Verification                                | 1-2  |
| Product Components   | 1-5  |
| Polyspace Products for C   | 1-5  |
| Polyspace Verification Environment                                 | 1-5  |
| Other Polyspace Components   | 1-9  |
| Other rolyspace components   | 1-9  |
| Installing Polyspace Products                                      | 1-11 |
|  | 1-11 |
| Obtaining Licenses for Polyspace <sup>®</sup> Client for C/C++ and |      |
|  | 1-11 |
|  |      |
|  | 1-12 |
|  | 1-12 |
| Tutorials in This Guide  | 1-13 |
| Additional Information and Support                                 | 1-15 |
|  | -    |
| 1  | 1-15 |
| MathWorks Online   | 1-15 |
| Related Products   | 1-16 |
|  | 1-16 |
|  | 1-16 |
|  | 1-16 |

| Overview                                |              | <br>            | ••• | ••  | <br>•••  | •••  | • |     | • |     | <br>•     |
|---|--------------|-----------------|-----|-----|----------|------|---|-----|---|-----|-----------|
| Example Files                           |              | <br>            | ••• | ••• | <br>••   |      |   |     | • |     |           |
|   |              |                 |     |     |          |      |   |     |   |     |           |
| Treating a New Pr                       | niect        |                 |     |     |          |      |   |     |   |     |           |
| Creating a New Pr<br>What Is a Project? | •            |                 |     |     |          |      |   |     |   |     |           |
| What Is a Project?                      | ?            | <br>            | ••• | ••• | <br>     | ••   | • | ••  | • |     | <br>•     |
|   | ?<br>Folders | <br><br>· · · · | ••• | ••• | <br><br> | <br> | • | ••• | • | ••• | <br><br>• |

## **Running a Verification**

| 8   | 8-2 |
|---|-----|
| Overview  | 3-2 |
| Before You Start 3                                      | 3-3 |
|   |     |
| Preparing for Verification                              | 8-4 |
|   | 8-4 |
|   | -5  |
|   | 6-6 |
|   |     |
| Launching Server Verification from Project              |     |
| Manager 3-:   | 10  |
| Starting the Verification 3-:                           | 10  |
| Monitoring Progress of the Verification 3-              | 13  |
| Removing Verification Results from the Server 3-        |     |
| Troubleshooting a Failed Verification 3-3               | -   |
|   | 20  |
|   | ~ ~ |
| Using Polyspace In One Click to Launch Verification 3-2 | -   |
| Overview of Polyspace In One Click 3-2                  | 23  |
| Setting the Active Project 3-2                          | 23  |
| Sending Files to Polyspace Software                     | 25  |

vi

2

#### Launching Client Verification from Project

| Manager  | 3-30          |
|--|---------------|
| Starting the Verification                      |               |
| Monitoring the Progress of the Verification    |               |
| Completing Verification                        |               |
| Stopping the Verification Before It is Complet | e <b>3-34</b> |

## **Reviewing Verification Results**

| Overview 4  | -2 |
|---|----|
|   | -2 |
|   | -  |
| Opening Verification Results 4-                   | -3 |
|   | -3 |
|   | -3 |
|   | -0 |
| Exploring Run-Time Checks Perspective 4-          | -4 |
|   | -4 |
|   | -6 |
|   | Ū  |
| Reviewing Results in Manual Mode                  | -8 |
|   | -8 |
|   | -8 |
|   | -8 |
| Reviewing Additional Examples of Checks 4-1       | 4  |
| Filtering Checks                                  |    |
|   |    |
| Reviewing Results in Assistant Mode 4-2           | 23 |
| What Is Assistant Mode? 4-2                       | 23 |
| Switching to Assistant Mode 4-2                   | 23 |
| Selecting the Methodology and Criterion Level 4-2 | 24 |
| Exploring Methodology for C 4-2                   | 24 |
| Reviewing Checks                                  |    |
| Defining a Custom Methodology 4-2                 |    |
|   | -0 |
| Automatically Testing Unproven Code 4-3           | 30 |

| Generating Reports of Verification Results | 4-31 |
|--|------|
| Polyspace Report Generator Overview        | 4-31 |
| Generating Report for example.c            | 4-32 |

# **Checking MISRA C Compliance**

# 5

| About Checking MISRA C Compliance Tutorial   | 5-2    |
|--|--------|
| Overview                                     | 5-2    |
| Before You Start                             | 5-2    |
| Setting Up MISRA C Checking                  | 5-3    |
| Opening Your Example Project                 | 5-3    |
| Creating New Verification                    | 5-4    |
| Setting MISRA C Checking Option              | 5-7    |
| Creating a MISRA C Rules File                | 5-8    |
| Excluding Files from the MISRA C Checking    | 5 - 11 |
| Configuring Text and XML Editors             | 5 - 12 |
| Saving the Project                           | 5-13   |
| Running a Verification with MISRA C Checking | 5-14   |
| Starting the Verification                    | 5 - 14 |
| Examining MISRA C Violations                 | 5-16   |
| Opening MISRA-C Report                       | 5-20   |
|  |        |

Index

# Introduction to Polyspace Products for Verifying C Code

- "Product Overview" on page 1-2
- "Product Components" on page 1-5
- "Installing Polyspace Products" on page 1-11
- "Working with Polyspace Software" on page 1-12
- "Additional Information and Support" on page 1-15
- "Related Products" on page 1-16

## **Product Overview**

#### In this section ...

"Overview of Polyspace Verification" on page 1-2

"The Value of Polyspace Verification" on page 1-2

## **Overview of Polyspace Verification**

Polyspace<sup>®</sup> products verify C, C++, and Ada code by detecting run-time errors before code is compiled and executed. Polyspace verification uses formal methods not only to detect errors, but to prove mathematically that certain classes of run-time errors do not exist.

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:

- "Ensure Software Reliability" on page 1-3
- "Decrease Development Time" on page 1-3
- "Improve the Development Process" on page 1-4

### **Ensure Software Reliability**

Polyspace software ensures the reliability of your C 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 know how much of your code is free of run-time errors, and you can improve the reliability of your code by fixing errors.

You can also improve the quality of your code by using Polyspace verification software to check that your code complies with MISRA  $C^{\circledast}$  standards.^1

### **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 during early coding phases allows you to find errors when it is less costly to fix them.

You use Polyspace software to verify C 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 using 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.

<sup>1.</sup> MISRA and MISRA C are registered trademarks of MISRA Ltd., held on behalf of the MISRA Consortium.

Using Polyspace verification software helps you to use your time effectively. Because you know which parts of your code are error-free, you can focus on the code that has definite errors or might have errors.

Reviewing 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 the 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 files 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 engineers 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 Products for C" on page 1-5

"Polyspace Verification Environment" on page 1-5

"Other Polyspace Components" on page 1-9

## **Polyspace Products for C**

The Polyspace products for verifying C code are combined with the Polyspace products for verifying C++ code. These products are:

- Polyspace<sup>®</sup> Client<sup>™</sup> for C/C++
- Polyspace<sup>®</sup> Server<sup>™</sup> for C/C++

Polyspace Client for C/C++ software is the management and visualization tool of Polyspace products. You use it to submit jobs for execution by the Polyspace Server, and to review verification results.

Polyspace Server for C/C++ software is the computational engine of Polyspace products. You use it to run jobs posted by Polyspace clients, and to manage multiple servers and queues.

## **Polyspace Verification Environment**

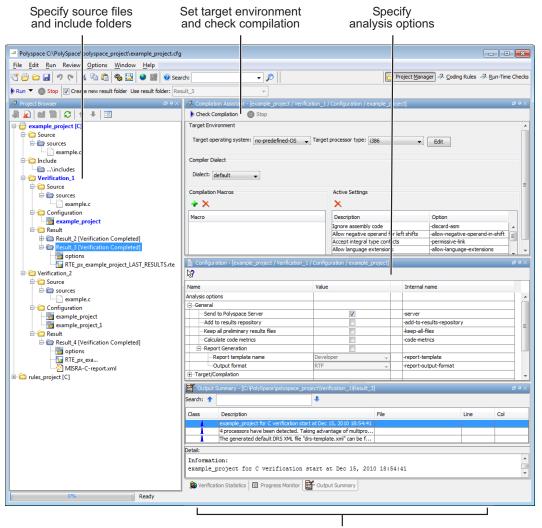
The Polyspace verification environment (PVE) is the graphical user interface of the Polyspace Client for C/C++ software. You use the Polyspace verification environment to create Polyspace projects, launch verifications, and review verification results.

The Polyspace verification environment consists of three perspectives:

- "Project Manager Perspective" on page 1-6
- "Coding Rules Perspective" on page 1-7
- "Run-Time Checks Perspective" on page 1-8

#### **Project Manager Perspective**

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



Monitor progress and view logs

You use the Project Manager perspective in the tutorial in Chapter 2, "Setting Up a Polyspace Project".

### **Coding Rules Perspective**

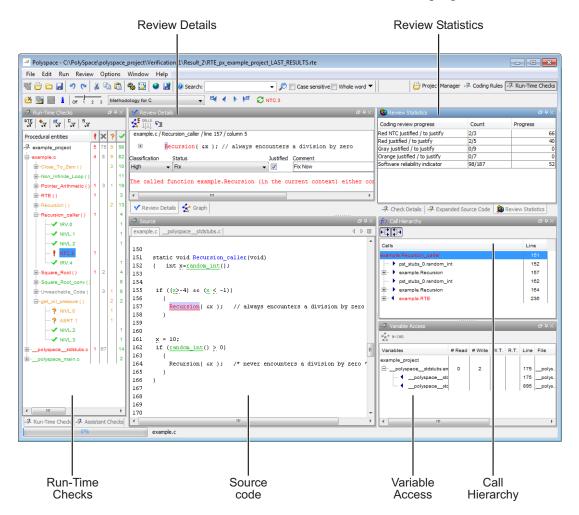
The Coding Rules perspective allows you to review results from the Polyspace coding rules checker, to ensure compliance with established coding standards.

| 🔫 Polys                 | pace - (       | C:\PolyS  | pace\po            | lyspace_pro        | oject∖Ve | rification_2\Re | esult_5\P | olySpac   | e-Doc\MISRA-C  | C-report.xml |              |                  |
|-------------------------|----------------|-----------|--------------------|--------------------|----------|-----------------|-----------|-----------|----------------|--------------|--------------|------------------|
| <u>F</u> ile <u>E</u> o | dit <u>R</u> u | in Rev    | iew <u>O</u> p     | otions <u>W</u> i  | ndow     | <u>H</u> elp    |           |           |                |              |              |                  |
| 📲 🗁 (                   | <u>&gt; </u>   | 90        | •   🔏 🗉            | à 🛍 🖗              | 1        | ۷ 🔛 🖌           |           |           |                |              |              | 🧢 Coding Rules 🔻 |
| <b>ĕ</b>                |                |           |                    |                    |          |                 |           |           |                |              |              |                  |
| 🔎 Assist                | ant Cod        | ing Rules | ; @ <del>P</del> × | A MISRA            | \ C      |                 |           |           |                |              |              | a t              |
| + +                     |                |           |                    | Filter             |          |                 | 📃 Hide    | justified | violated rules |              |              | i                |
| Rule                    | File           | Line      | Col                | w/e                | Rule     | File            | Line      | Col       | Classification | Status       | Justified    | Comment          |
|                         |                |           |                    | error              | 16.3     | include.h       | 33        |           |                |              |              |                  |
|                         |                |           |                    | warning            | 17.4     | example.c       | 97        | -         |                |              |              |                  |
|                         |                |           |                    | warning<br>warning | 17.4     | example.c       | 114       |           |                |              |              |                  |
|                         |                |           |                    | warning            | 17.4     | example.c       | 110       | 14        |                |              |              |                  |
|                         |                |           |                    |                    |          |                 |           |           |                |              |              |                  |
| 🧢 Rule o                | letails        |           |                    |                    |          |                 |           |           |                |              |              | ₽₽×              |
| Rule: 1                 | 16.3 (         | error)    | : Ident            | tifiers s          | hall h   | be given fo     | r all     | of the    | parameters     | in a funct:  | ion prototyp | e declaration.   |
| File: (                 | :\Pol          | ySpace    | \polysp            | pace_proj          | ect\in   | ncludes\inc     | lude.h    | line :    | 33 (column 2   | 8)           |              |                  |
| Source<br> exter        |                | d Exec    | _One_C             | ycle (int<br>^     | ;);      |                 |           |           |                |              |              |                  |
|                         |                | 0         | %                  |                    | F        | Ready           |           |           |                |              |              |                  |

You use the Coding Rules perspective in the tutorial in Chapter 5, "Checking MISRA C Compliance".

#### **Run-Time Checks Perspective**

The Run-Time Checks perspective allows you to review verification results, comment individual checks, and track review progress.



You use the Run-Time Checks perspective in the tutorial in Chapter 4, "Reviewing Verification Results".

## **Other Polyspace Components**

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

- Polyspace Queue Manager Interface (Spooler)
- Polyspace in One Click
- 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 C/C++ 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.

| 🔛 Polyspace Queue Manager Interface |             |                 |   |          |           |                       |          |
|-------------------------------------|-------------|-----------------|---|----------|-----------|-----------------------|----------|
| Operations <u>H</u> elp             |             |                 |   |          |           |                       |          |
| ID                                  | Author      | Application     | Results folder  | CPU      | Status    | Date                  | Language |
| 23                                  | username    | example_project | C:\PolySpace\polyspace_project\Verification_1\Result_9  | AH-SRUNS | completed | 15-Dec-2010, 16:03:33 | с        |
| 24                                  | username    | example_project | C:\PolySpace\polyspace_project\Verification_1\Result_10 | AH-SRUNS | running   | 15-Dec-2010, 16:03:33 | С        |
| 25                                  | username    | example_project | C:\PolySpace\polyspace_project\Verification_1\Result_11 |          | queued    | 15-Dec-2010, 16:03:21 | С        |
|                                     |             |                 |   |          |           |                       |          |
|                                     |             |                 |   |          |           |                       |          |
|                                     |             |                 |   |          |           |                       |          |
| Conne                               | cted to Que | ue Manager AH-  | SRUNSTRO.dhcp.mathworks.com                             |          |           | U                     | ser mode |

You use the Polyspace Queue Manager in the tutorial "Launching Server Verification from Project Manager" on page 3-10.

### **Polyspace in One Click**

Polyspace in One Click is a convenient way to verify multiple files using the same set of options.

After creating a project with the options that you want, you can use Polyspace in One Click to designate that project as the *active project*, and then send source files to Polyspace software for verification with a single mouse click.

You use Polyspace in One Click in the tutorial "Using Polyspace In One Click to Launch Verification" on page 3-23.

#### **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, coding rule violations, and run-time checks through the lifecycle of a project.

For information on using Polyspace Metrics, see "Software Quality with Polyspace Metrics" in the *Polyspace Products for C User's Guide*.

# **Installing Polyspace Products**

#### In this section...

"Finding the Installation Instructions" on page 1-11

"Obtaining Licenses for Polyspace" Client for C/C++ and Polyspace" Server for C/C++" on page 1-11

## Finding the Installation Instructions

The tutorials in this guide require Polyspace Client for C/C++ and Polyspace Server for C/C++. Instructions for installing Polyspace products are in the *Polyspace Installation Guide*. Before installing Polyspace products, you must obtain the necessary licenses.

## Obtaining Licenses for Polyspace Client for C/C++ and Polyspace Server for C/C++

For information about obtaining licenses for Polyspace products, see "Polyspace License Installation" in the *Polyspace Installation Guide*.

# Working with Polyspace Software

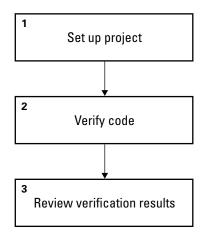
#### In this section ...

"Basic Workflow" on page 1-12

"Tutorials in This Guide" on page 1-13

## **Basic Workflow**

The following graphic shows the basic workflow for using Polyspace software to verify C source code.



In this workflow, you:

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

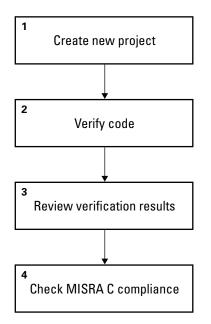
You can use the Project Manager perspective to start the verification or you can select files from a Microsoft<sup>®</sup> Windows<sup>®</sup> folder and send them to Polyspace software for verification. For verifications that run on a server, you use the Polyspace Queue Manager Interface (Polyspace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

manage the verification and download the results to a client. You can set an option to check MISRA C compliance in the first stage of the verification.<sup>2</sup>

**3** Use the Run-Time Checks perspective to review verification results.

## **Tutorials in This Guide**

The tutorials guide you through the basic workflow, including the different options for running verifications. The following graphic shows the workflow you follow in these tutorials.



In this workflow, you:

1 Create a new project that you use for the workflow.

This step is in the tutorial in Chapter 2, "Setting Up a Polyspace Project".

**2** Verify a single C file.

<sup>2.</sup> MISRA and MISRA C are registered trademarks of MISRA Ltd., held on behalf of the MISRA Consortium.

This step is in the tutorial in Chapter 3, "Running a Verification". In this tutorial, you verify the same file using three different methods of running a verification:

- Start a verification that runs on a server using the Project Manager perspective.
- Send files to a server for verification using Polyspace In One Click.
- Start a verification that runs on a client using the Project Manager perspective.
- **3** Review the verification results.

This step is in the tutorial in Chapter 4, "Reviewing Verification Results".

**4** Modify the project to include MISRA C checking and review the MISRA C violations in the example file.

This step is in the tutorial in Chapter 5, "Checking MISRA C Compliance".

# **Additional Information and Support**

In this section...

"Product Help" on page 1-15

"MathWorks Online" on page 1-15

## **Product Help**

To access Polyspace online Help, select **Help > Help**.

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

/www.mathworks.com/access/helpdesk/help/toolbox/polyspace/polyspace.html

## **MathWorks Online**

For additional information and support, go to:

www.mathworks.com/products/polyspace

# **Related Products**

#### In this section ...

"Polyspace Products for Verifying C++ Code" on page 1-16

"Polyspace Products for Verifying Ada Code" on page 1-16

"Polyspace Products for Linking to Models" on page 1-16

## Polyspace Products for Verifying C++ Code

For information about Polyspace products that verify C++ code, go to:

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

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

## **Polyspace Products for Verifying Ada Code**

For information about Polyspace products that verify Ada code, go to:

http://www.mathworks.com/products/polyspaceclientada/

http://www.mathworks.com/products/polyspaceserverada/

## **Polyspace Products for Linking to Models**

For information about Polyspace products that link to models, go to:

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

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

# Setting Up a Polyspace Project

- "About Setting Up a Project Tutorial" on page 2-2
- "Creating a New Project" on page 2-3

# **About Setting Up a Project Tutorial**

#### In this section...

"Overview" on page 2-2

"Example Files" on page 2-2

## Overview

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

## **Example Files**

This tutorial uses the source file example.c that comes with the installation. You learn more about the files and folders required for this tutorial in "Preparing Project Folders" on page 2-4.

# **Creating a New Project**

#### In this section ...

"What Is a Project?" on page 2-3

"Preparing Project Folders" on page 2-4

"Opening Polyspace Verification Environment" on page 2-5

"Creating a New Project to Verify the Example C File" on page 2-7

## 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
- One or more configurations, specifying a set of analysis options
- One or more verifications, 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 project. You 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 a C file with Polyspace software, you must know the locations of the C source file and the include files. 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 in that folder, create separate folders for the source files, include files, and results.

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
- 3 Copy the file example.c from

Polyspace\_Install\Examples\Demo\_C\_Single-File\sources

to

polyspace\_project\sources

Polyspace\_Install is the installation folder.

4 Copy the files include.h and math.h from

Polyspace\_Install\Examples\Demo\_C\_Single-File\sources

to

polyspace\_project\includes.

## **Opening Polyspace Verification Environment**

You 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 (Windows systems).



**Note** On a Linux<sup> $\mathbb{R}$ </sup> or UNIX<sup> $\mathbb{R}$ </sup> system, use the following command:

/usr/local/Polyspace/PVE/bin/polyspace

**2** If you have only Polyspace Client for C/C++ software installed on your computer, skip this step. 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.



#### **3** Select **Polyspace for C/C++** and click **OK**.

The Polyspace Verification Environment opens.

|   |                                    | environment<br>compilation   |            | Spe<br>analysis                | ecify<br>s optior    | IS                     |                  |
|---|------------------------------------|--|------------|--------------------------------|----------------------|------------------------|------------------|
| Polyspace C:\PolySpace project\example_project.cf                               | J                                  |  |            |                                |                      |                        | - • •            |
| File Edit Run Review Options Window Help  |                                    |  |            |                                |                      |                        |                  |
| T 🕒 🕞 🚽 🔿 🗞 🕼 🛍 🆓 🔛 🔍 📰 🖉 See   | rchi                               | - <b></b>  |            | 0                              | Project Mana         | ger 🥕 Coding Rules 🦂   | Run-Time Checks  |
|   |                                    |  |            | 16                             | Trojece <u>m</u> ana |                        |                  |
| Run 🔻 🕘 Stop 📝 Crez is new result folder Use result folder: Re                  | sult_3                             | -  |            |                                |                      |                        |                  |
| 🐥 Project Browser 🗇 무 🗙   | 🛹 Compilation Assista              | it - [example_project / Verification                                     | 1_1 / Conf | iguration / example_           | p oject]             |                        | ē ₽×             |
| 🗐 🔊 🖆 🛅 😂 🕂 🕂 🖾   | Check Compilation                  | 🚳 Stop   |            |                                |                      |                        |                  |
| example_project [C]   | Target Environment                 |  |            |                                |                      |                        | <b>^</b>         |
| 🖨 🗁 Source  | Target operating sys               | tem: no-predefined-OS 🖉 Ta   | raet proce | ssor type: 1296                |                      | Edit                   |                  |
| example.c   |                                    |  | 3          | 1000                           | · · · ·              | Cuit                   |                  |
| example.c   | Compiler Dialect                   |  |            |                                |                      |                        |                  |
| includes  |                                    |  |            |                                |                      |                        |                  |
| 🖨 🗁 Verification_1  | Dialect: default                   | <b>~</b>   |            |                                |                      |                        |                  |
| De Conce  | Compilation Macros                 |  | Acti       | ve Settings                    |                      |                        | =                |
| example.c   | <b>+</b> ×                         |  | ×          | -                              |                      |                        |                  |
|   |                                    |  |            |                                |                      |                        |                  |
| example_project   | Macro                              |  |            | scription<br>ore assembly code |                      | Option<br>-discard-asm |                  |
| 🖻 🗁 Result  |                                    |  |            | w negative operand             | f r left shifts      | -allow-negative-operan | d-in-shift       |
| Result_2 [Verification Completed]     General Result_3 [Verification Completed] |                                    |  |            | ept integral type con          |                      | -permissive-link       |                  |
| options   |                                    |  | Allo       | w language extension           | 3                    | -allow-language-extens | ions 🕌 👻         |
| RTE_px_example_project_LAST_RESULTS.rte   | Configuration - [exa               | ample_project / Verification_1 / Co                                      | nfiguratio | n / example_project]           |                      |                        | 0 4 ×            |
| - Cerification_2  | 127                                |  |            |                                | 1                    |                        |                  |
| □ - Cource<br>□ - Cources   | Name                               | V  | alue       |                                | Internal r           | name                   |                  |
| example.c   | Analysis options                   |  |            |                                |                      |                        | A                |
| Configuration   | General                            |  |            | _                              |                      |                        |                  |
|   | Send to Polyspa                    |  |            |                                | -server              | sults-repository       |                  |
| example_project_1   | Keep all prelimin                  |  |            |                                | -add-to-re           |                        | =====            |
| ⊟-  | Calculate code                     |  |            |                                | -code-met            |                        |                  |
| options   | Report Generat                     |  |            |                                |                      |                        |                  |
| TE_px_exa   | Report tem                         |  | eveloper   | T                              | -report-te           |                        |                  |
| MISRA-C-report.xml  | Output form<br>⊡Target/Compilation | nat R1   | IF         | Ŧ                              | -report-ou           | itput-format           |                  |
|   |                                    |  |            |                                |                      |                        | +                |
|   |                                    | [C:\PolySpace\polyspace_project  |            | n_1\Result_3]                  |                      |                        | 0 <del>9</del> × |
|   | Search: 🕇                          | +  |            |                                |                      |                        |                  |
|   | Class Descript                     | ion  |            | File                           |                      | Line                   | Col              |
|   |                                    | project for C verification start at I                                    |            |                                |                      |                        |                  |
|   |                                    | sors have been detected. Taking a<br>erated default DRS XML file "drs-te |            |                                |                      |                        |                  |
|   | Detail:                            |  | npiaterxii |                                |                      |                        |                  |
|   | Information:                       |  |            |                                |                      |                        |                  |
|   |                                    | for C verification star  | t at De    | c 15, 2010 18:                 | 54:41                |                        |                  |
|   |                                    |  |            |                                |                      |                        | •                |
|   | Verification Statist               | ics 🛛 Progress Monitor 🖉 C   | output Sur | nmary                          |                      |                        |                  |
| 0% Ready  |                                    |  |            |                                |                      |                        |                  |
|   |                                    |  |            |                                |                      |                        |                  |
|   |                                    |  |            |                                |                      |                        |                  |

Monitor progress and view logs

By default, the Polyspace Verification Environment displays the Project Manager perspective. The Project Manager perspective has three main panes.

| Use this section                | For  |
|---------------------------------|--|
| Project Browser<br>(upper-left) | Specifying:<br>• Source files  |
|                                 | • Include folders  |
|                                 | Results folder   |
| Configuration<br>(upper-right)  | Specifying analysis options  |
| Output<br>(lower-right)         | Monitoring the progress of a verification, and viewing<br>status, log messages, and general verification statistics. |

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

## Creating a New Project to Verify the Example C File

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 for verifying example.c.

You create a new project by:

- "Opening a New Project" on page 2-7
- "Specifying Source Files and Include Folders" on page 2-11
- "Specifying Target Environment" on page 2-12
- "Specifying Analysis Options" on page 2-13
- "Saving the Project" on page 2-14

### **Opening a New Project**

To open a new project for verifying example.c:

#### 1 Select File > New Project.

The Polyspace Project – Properties dialog box opens:

| Polyspace Project - Properties  |                                      |  |  |  |  |  |
|---------------------------------|--------------------------------------|--|--|--|--|--|
| Project definition and location |                                      |  |  |  |  |  |
| Project name:                   | Project name: example_project        |  |  |  |  |  |
| Version:                        | 1.0                                  |  |  |  |  |  |
| Author:                         | username                             |  |  |  |  |  |
| Default location                |                                      |  |  |  |  |  |
| Location: C:\Poly               | cation: C:\PolySpace\example_project |  |  |  |  |  |
| Project language                |                                      |  |  |  |  |  |
| O                               |                                      |  |  |  |  |  |
| © C++                           |                                      |  |  |  |  |  |
|                                 |                                      |  |  |  |  |  |
|                                 |                                      |  |  |  |  |  |
|                                 | Back Next Finish Cancel              |  |  |  |  |  |

- 2 In the **Project name** field, enter example\_project.
- **3** Clear the **Default location** check box.

**Note** Clearing the **Default location** check box allows you to specify the location of your project files. In this tutorial, you change the default location to the project folder that you created in "Preparing Project Folders" on page 2-4. Changing the default location makes it easier to specify source files and include folders.

**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 In the Project language section, select C.
- 6 Click Finish.

The example\_project opens in the Polyspace verification environment.

| A Polyspace   |   |   |  |  |  |  |  |
|---|---|---|--|--|--|--|--|
| File Edit Run Review Options Window Help                |   |   |  |  |  |  |  |
|   | 🥘 Search: 🗸 🔎   | Project Mana  | ger 🗦 Coding Rules 🦩 Run-Time Checks           |  |  |  |  |
| Run 🔻 🕲 Stop 💟 Create new result folder Use result fold |   |   |  |  |  |  |  |
|   |   |   |  |  |  |  |  |
| - ♣ Project Browser                                     | Compilation Assistant - [example_project / Verification_1 / C     | onfiguration / example_projectj                             | 리 무 ×  |  |  |  |  |
| 🗛 🔊 😅 🕲 😂 + + 🖾   | Check Compilation 🚳 Stop  |   |  |  |  |  |  |
| 🖃 🚰 example_project [C]                                 | Target Environment  |   |  |  |  |  |  |
| Source  | Target operating system: Linux  Target processor type: 386  Edit  |   |  |  |  |  |  |
| Include   |   |   |  |  |  |  |  |
| Source  | Compiler Dialect  |   |  |  |  |  |  |
| Configuration   | Dialect: default  |   |  |  |  |  |  |
| Result  | Compilation Macros  | Active Settings   |  |  |  |  |  |
|   | + ×   | ×   |  |  |  |  |  |
|   | Macro   | Description   | Option   |  |  |  |  |
|   |   | Ignore assembly code  | -discard-asm                                   |  |  |  |  |
|   |   | Allow negative operand for left shifts                      | -allow-negative-operand-in-shift               |  |  |  |  |
|   |   | Accept integral type conflicts<br>Allow language extensions | -permissive-link<br>-allow-language-extensions |  |  |  |  |
|   |   | Allow non int types for bitfields                           | -allow-non-int-bitfield                        |  |  |  |  |
|   |   | Allow anonymous unions/structure fields                     | -allow-unnamed-fields                          |  |  |  |  |
|   |   | Code from DOS or Windows filesystem                         | -dos   |  |  |  |  |
|   |   | Allow undefined global variables                            | -allow-undef-variables                         |  |  |  |  |
|   | Configuration - [example_project / Verification_1 / Configuration | tion ( annual anni all                                      | 고 무 ×  |  |  |  |  |
|   | Configuration - [example_project / vertication_1 / configuration] | uon / example_projectj                                      | × + ⊔  |  |  |  |  |
|   | Name  | alue  | Internal name                                  |  |  |  |  |
|   | Analysis options  | auc   |  |  |  |  |  |
|   | General   |   | ^  |  |  |  |  |
|   | Send to Polyspace Server  |   |  |  |  |  |  |
|   | -Add to results repository  |   | -server<br>-add-to-results-repository          |  |  |  |  |
|   | Keep all preliminary results files                                |   | -keep-all-files                                |  |  |  |  |
|   |   |   | -code-metrics                                  |  |  |  |  |
|   |   |   | E  |  |  |  |  |
|   |   | eveloper v  | -report-template                               |  |  |  |  |
|   |   | TF *  | -report-output-format                          |  |  |  |  |
|   | Target/Compilation  | *   |  |  |  |  |  |
|   | Compliance with standards   |   |  |  |  |  |  |
|   | Polyspace inner settings  |   |  |  |  |  |  |
|   | Precision/Scaling   |   |  |  |  |  |  |
|   | ·   |   |  |  |  |  |  |
|   | 1=32  |   | a ex   |  |  |  |  |
|   | Output Summary  |   | B <sup>+</sup> ₩ ×                             |  |  |  |  |
|   |   |   |  |  |  |  |  |
|   |   |   |  |  |  |  |  |
|   |   |   |  |  |  |  |  |
|   |   |   |  |  |  |  |  |
|   |   |   |  |  |  |  |  |
|   |   |   |  |  |  |  |  |
|   |   |   |  |  |  |  |  |
|   |   |   |  |  |  |  |  |
|   | 📝 Full Log 🛕 Verification Statistics 🗏 Progress Monitor           | Cutput Summary  |  |  |  |  |  |
| 0% Ready  |   |   |  |  |  |  |  |
| Ready   |   |   |  |  |  |  |  |

#### **Specifying Source Files and Include Folders**

To specify the source files and include folders for the verification of example.c:

- **1** In the Project Browser, select the Source folder.
- 2 Click the Add source icon in the upper left the Project Browser.

The Polyspace Project – Add Source Files and Include Folders dialog box opens.

| 🜁 Polyspace Project - Add Source Files and Include Folders 🧮 |  |                              |   |            |                         |      |  |  |
|--|--|------------------------------|---|------------|-------------------------|------|--|--|
| Look in:   | la polyspace                                 | project                      | • | 🏂 📂 🖽 📟    | 📝 Add recursively 📓 🕇 🕈 |      |  |  |
| Recent Items<br>Desktop<br>Documents<br>Computer             | includes<br>sources                          |                              |   |            | example_project [C]     |      |  |  |
| Network  | File <u>n</u> ame:<br>Files of <u>t</u> ype: | includes<br>(*,c) files only | • | Add Source |                         |      |  |  |
|  |  |                              |   |            | Back Next Finish Ca     | ncel |  |  |

- **3** The project folder polyspace\_project should appear in Look in. If it does not, navigate to that folder.
- 4 Select the sources folder, then click Add Source.

The example.c file appears in the Source tree for example\_project.

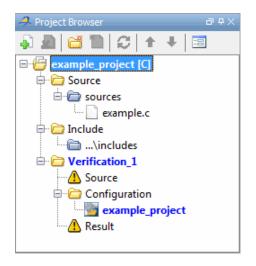
5 Select the includes folder, then click Add Include.

The includes folder appears in the Include tree for example\_project.

**Note** In addition to the include folders you specify, Polyspace software automatically adds the standard includes to your project.

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

The Project Browser now looks like the following graphic.



#### **Specifying Target Environment**

Many applications are designed to run on specific target CPUs and operating systems. Since some run-time errors are dependent on the target, you must specify the type of CPU and operating system used in the target environment before running a verification.

The Compilation Assistant window in the top-right section of the Project Manager perspective allows you to specify the target operating system and processor type for your application. To specify the target environment for this tutorial:

1 In the **Target operating system** drop-down menu, select no\_predefined\_OS.

| A Compilation Assistant - [example_project / Verification_1 / Configuration / example_project] | 0 <del>4</del> × |
|--|------------------|
| Check Compilation Stop   |                  |
| - Target Environment   |                  |
| Target operating system: no-predefined-OS - Target processor type: 1386                        | Edit             |
| Compiler Dialect   |                  |
| Dialect: default   |                  |

2 In the Target processor type drop down menu, select i386.

For more information about emulating your target environment, see "Setting Up a Target" in the *Polyspace Products for C User's Guide*.

#### **Specifying Analysis Options**

The Configuration window in the middle-right section of the Project Manager perspective allows you to set Analysis options that Polyspace software uses during the verification process.

For this tutorial, you should use the default values for all options.

| Name                                   | Value       | Internal name              |
|--|-------------|----------------------------|
| Analysis options                       |             |                            |
| 🚊 General                              |             |                            |
| Send to Polyspace Server               |             | -server                    |
| Add to results repository              |             | -add-to-results-repository |
| ····Keep all preliminary results files |             | -keep-all-files            |
| ····Calculate code metrics             |             | -code-metrics              |
| - Report Generation                    |             |                            |
| Report template name                   | Developer 🚽 | -report-template           |
| Output format                          | RTF 👻       | -report-output-format      |
|  |             |                            |
| Compliance with standards              |             |                            |
| ⊕ Polyspace inner settings             |             |                            |
| ±Precision/Scaling                     |             |                            |
|  |             |                            |

For more information about analysis options, see "Options Description" in the *Polyspace Products for C Reference*.

#### **Saving the Project**

To save the project, select **File > Save**.

Polyspace software saves your project using the Project name and Location you specified when creating the project.

# Running a Verification

- "About Running a Verification Tutorial" on page 3-2
- "Preparing for Verification" on page 3-4
- "Launching Server Verification from Project Manager" on page 3-10
- "Using Polyspace In One Click to Launch Verification" on page 3-23
- "Launching Client Verification from Project Manager" on page 3-30

# **About Running a Verification Tutorial**

#### In this section...

"Overview" on page 3-2

"Before You Start" on page 3-3

## Overview

Once you have created the project example.cfg, as described in "Creating a New Project" on page 2-3, you can run the verification.

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

| Use    | For   |
|--------|---|
| Server | Best performance  |
|        | • Large files (more than 800 lines of code, including comments)   |
|        | • Multitasking  |
| Client | • An alternative to the server when the server is busy  |
|        | • Small files with no multitasking  |
|        |   |
|        | <b>Note</b> Verification on a client takes more time. You might not be able to use your client computer when a verification is running on it. |
|        |   |

You can start a verification using either the Project Manager or Polyspace In One Click. With either method, the verification can run on a server or a client.

| Use                    | For  |
|------------------------|--|
| Project Manager        | A basic way to start a verification.   |
|                        | You specify the source files in the project file.<br>With the project open, you click a button to start<br>the verification.   |
| Polyspace In One Click | A convenient way to start the verification of<br>several files which use the same verification<br>options.   |
|                        | Once you specify the project file containing the<br>verification options, you specify the source files<br>by selecting them from a Microsoft Windows<br>folder. You start the verification by sending the<br>selected files to Polyspace software. |

In this tutorial, you learn how to run a verification on a server and on a client, and how to start a verification using the Project Manager and Polyspace In One Click. You verify the file example.c three times using a different method each time. You use:

- Project Manager to start a verification that runs on a server.
- Polyspace In One Click to start a verification that runs on a server.
- Project Manager to start a verification that runs on a client.

Each verification stores the same results in your project. You review these results in the tutorial Chapter 4, "Reviewing Verification Results".

## **Before You Start**

Before you start this tutorial, you must complete Chapter 2, "Setting Up a Polyspace Project". You use the folders and project file, example.cfg, from that tutorial.

# **Preparing for Verification**

#### In this section ...

"Opening the Project" on page 3-4

"Specifying Source Files to Verify" on page 3-5

"Checking for Compilation Problems" on page 3-6

## **Opening the Project**

To run a verification, you must have an open project file. For this tutorial, you use the project file example.cfg that you created in Chapter 2, "Setting Up a Polyspace Project". If example\_project.cfg is not already open, open it.

To open example\_project.cfg:

- 1 If the Polyspace software is not already open, open it.
- 2 Select File > Open Project.

The Open a Polyspace project file dialog box opens.

- 3 In the Look in drop-down list box, 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 verifications. Each of these verifications can analyze a specific set of source files using a specific set of analysis options.

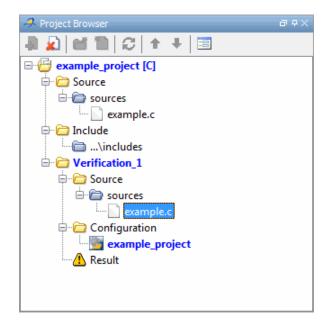
Therefore, before you launch a verification, you must specify which files in your project that you want to verify. In the example\_project in this tutorial, there is only one file to verify.

To copy source files to a verification:

1 In the Project Browser Source tree, right click example.c.

#### **2** Select Copy Source File to > Verification\_(1).

The example.c file appears in the Source tree of Verification\_(1).

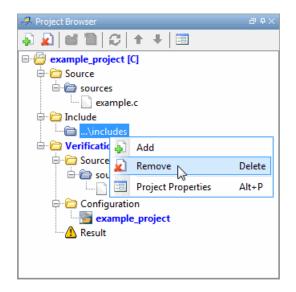


## **Checking for Compilation Problems**

The Compilation Assistant allows you to check your project for compilation problems before launching a verification. When the Compilation Assistant detects an error, it reports the problem and suggests possible solutions.

To check your project for compilation problems:

1 In the Project Browser Source tree, right click the Include folder (...\includes), then select **Remove**. This will cause a compilation error.



2 In the Compilation Assistant window, click Check Compilation.

| Compilation Assistant - [example_project / Verificat | ion_1 / Configuration / example_project] | 9 <del>9</del> X                 |
|--|--|----------------------------------|
| Check Compilation Stop                               |  |                                  |
| Target Environment 🕏                                 |  |                                  |
| Target operating system: no-predefined-OS            | Target processor type: [i386             | ✓ Edit                           |
| Compiler Dialect                                     |  |                                  |
| Dialect: default                                     |  |                                  |
| Compilation Macros                                   | Active Settings                          |                                  |
| + ×  | ×  |                                  |
| Macro  | Description                              | Option                           |
|  | Ignore assembly code                     | -discard-asm                     |
|  | Allow negative operand for left shifts   | -allow-negative-operand-in-shift |
|  | Accept integral type conflicts           | -permissive-link                 |
|  | Allow language extensions                | -allow-language-extensions       |
|  | Allow non int types for bitfields        | -allow-non-int-bitfield          |
|  | Allow anonymous unions/structure fields  | -allow-unnamed-fields            |
|  | Code from DOS or Windows filesystem      | -dos                             |
|  | Allow undefined global variables         | -allow-undef-variables           |

The software compiles your code and checks for errors, and reports the results in the Output Summary.

| Туре | File Line Message Suggestion/Remark |   |  |                                   | Action |     |
|------|-------------------------------------|---|--|-----------------------------------|--------|-----|
| 1    | example.c                           | 1 | could not open source file "math.h"    | Add include folder for: math.h    | -      | Add |
| 1    | example.c                           | 2 | could not open source file "include.h" | Add include folder for: include.h | -      | Add |

Because you removed the include folder, the software reports a compilation error for the project, along with suggested solutions for the problem.

**3** Select the Suggestion/Remark column to see a list of possible solutions for the problem.

| <b>2</b> 0                                      | utput Summa | ary - [C | :\Users\srunstro\AppData\Local\Temp\F  | olyspace\1298060142138\CompileAssista | nt] ♂₽× |
|---|-------------|----------|--|---------------------------------------|---------|
| Preprocessing Errors: 2     Filter warnings (0) |             |          |  |                                       |         |
| Туре  | File        | Line     | Message                                | Suggestion/Remark                     | Action  |
| 1   | example.c   | 1        | could not open source file "math.h"    | Add include folder for: math.h        | Add     |
| 1   | example.c   | 2        | could not open source file "include.h" | Add include folder for: include.h     | Add     |
|   |             |          |  | Add include folder for: include.h     |         |
|   |             |          |  | Set option: -ignore-missing-headers   |         |

In this case, you can either add the missing include file, or set an option that will attempt to compile the code without the missing include file.

- 4 Select Set option: -ignore-missing-headers, then click Apply.
- **5** The software automatically sets the option **Ignore missing header files** for your project, and displays the option in the Compilation Assistant Active Settings table.

| Active Settings                         |                                  |  |  |  |  |
|---|----------------------------------|--|--|--|--|
| ×                                       |                                  |  |  |  |  |
| Description                             | Option                           |  |  |  |  |
| Ignore assembly code                    | -discard-asm                     |  |  |  |  |
| Allow negative operand for left shifts  | -allow-negative-operand-in-shift |  |  |  |  |
| Accept integral type conflicts          | -permissive-link                 |  |  |  |  |
| Allow language extensions               | -allow-language-extensions       |  |  |  |  |
| Allow non int types for bitfields       | -allow-non-int-bitfield          |  |  |  |  |
| Allow anonymous unions/structure fields | -allow-unnamed-fields            |  |  |  |  |
| Ignore missing header files             | -ignore-missing-headers          |  |  |  |  |
| Code from DOS or Windows filesystem     | -dos                             |  |  |  |  |
| Allow undefined global variables        | -allow-undef-variables           |  |  |  |  |

6 Select Check Compilation to check your project again.

The same errors appear, since the code cannot be compiled without include.h.

7 In the Output Summary window, select Add include folder for: include.h, then click Add.

The Add Source Files and Include Folders dialog box opens.

| 🐣 Polyspace Proj   | ect - Add Sourc     | e Files and Include Folders |   |             |  |
|--|---------------------|-----------------------------|---|-------------|--|
| Look in:   | polyspace_          | project                     | • | € 🔝 🔁 🏂     | 📝 Add recursively 📓 🕇 🕈  |
| Recent Items<br>Desktop<br>Documents<br>Computer<br>Computer | includes<br>sources | indudes                     |   | Add Sgurce  | example_project [C]<br>example_c<br>example.c<br>Include<br>includes |
|  | Files of type:      | (*.c) files only            | • | Add Include |  |
|  |                     |                             |   |             | Back Next Finish Cancel  |

- 8 If necessary, navigate to the polyspace\_project folder.
- **9** Select the includes folder, then click **Add Include**.

The includes folder appears in the Include tree for example\_project.

- 10 Click Finish.
- **11** Select **Check Compilation** in the Compilation Assistant to check your project again.

The message  ${\tt Compilation}$  succeeded appears in the Output Summary.

| i] ∂₽× |
|--------|
|        |
| Action |
|        |
|        |
|        |

# Launching Server Verification from Project Manager

#### In this section ...

"Starting the Verification" on page 3-10

"Monitoring Progress of the Verification" on page 3-13

"Removing Verification Results from the Server" on page 3-19

"Troubleshooting a Failed Verification" on page 3-20

## **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** Select the **Send to Polyspace Server** check box in the General Analysis options.

| Name                                   | Value     | Internal name              |
|--|-----------|----------------------------|
| Analysis options                       |           |                            |
| General                                |           |                            |
| Send to Polyspace Server               | <b>V</b>  | -server                    |
| Add to results repository              |           | -add-to-results-repository |
| ····Keep all preliminary results files |           | -keep-all-files            |
| ·····Calculate code metrics            |           | -code-metrics              |
| -Report Generation                     |           |                            |
| Report template name                   | Developer | <br>-report-template       |
| Output format                          | RTF       | <br>-report-output-format  |
|  |           |                            |
| ⊕Compliance with standards             |           |                            |
| ⊕Polyspace inner settings              |           |                            |
|  |           |                            |
|  |           |                            |

**2** Click the **Run** button **Run** on the Project Manager toolbar.

**Note** If you see the message Verification process failed, click **OK** and go to "Troubleshooting a Failed Verification" on page 3-20.

The verification has three main phases:

- **a** Checking syntax and semantics (the compile phase). Because Polyspace software is independent of any particular C compiler, it ensures that your code is portable, maintainable, and complies with ANSI<sup>®</sup> standards.
- **b** Generating a main if the Polyspace software does not find a main and you have selected the **Generate a Main** option. For more information about generating a main, see "Main Generator Behavior for Polyspace Software" in the *Polyspace Products for C Reference*.

**c** Analyzing the code for run-time errors and generating color-coded results.

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 the Output Summary window gives you the identification number (Analysis ID) for the verification. For this verification, the identification number is 1.

| 📲 Output S | Summary - [C: \PolySpace\polyspace_project\Verification_1\Result_1]   |      |      | ð   |   |
|------------|---|------|------|-----|---|
| Search: 🛧  | +   |      |      |     |   |
| Class      | Description   | File | Line | Col |   |
| 1          | example_project for C verification start at Dec 15, 2010 15:57:15   |      |      |     |   |
| 1          | The generated default DRS XML file "drs-template.xml" can be found in <result_dir< td=""><td></td><td></td><td></td><td></td></result_dir<> |      |      |     |   |
| 1          | Analysis ID : 1   |      |      |     | - |

**3** For information on any message in the log, click the message.

## **Monitoring 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.

|   | Progress Monitor - [C: | \PolySpace\polyspace_pro | ject\Verification_1\Result_ | _1]                    | 0 <del>4</del> X |
|---|------------------------|--------------------------|-----------------------------|------------------------|------------------|
|   |                        |                          | Verification is running     | g on server with ID: 1 |                  |
|   | Intermediate: 100%     | Level0: 100%             | Level 1: 89%                | Level2 : 0%            | Level3 : 0%      |
|   | 00:00:03               | 00:00:20                 | 00:00:13                    | 00:00:00               | 00:00:00         |
|   |                        |                          |                             |                        |                  |
|   |                        |                          |                             |                        |                  |
| 4 |                        | III                      |                             |                        | 4                |

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. Follow the next steps to view the logs:

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

- **3** Click the **Refresh** button to update the display as the verification progresses.
- **4** Click the **Full Log** tab to display messages, errors, and statistics for all phases of the verification.

**Note** You can search the logs. In the **Search in the log** box, enter a search term and click the left arrows to search backward or the right arrows 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:

1 Double-click the **Polyspace Spooler** icon on the desktop.



The Polyspace Queue Manager Interface opens.

| 🚰 Polyspace Queue Manager Interface 📃 💷 🗾 |            |                 |   |             |           |                       |           |
|---|------------|-----------------|---|-------------|-----------|-----------------------|-----------|
| Operations Help                           |            |                 |   |             |           |                       |           |
| ID  | Author     | Application     | Results folder  | CPU         | Status    | Date                  | Language  |
| 23  | username   | example_project | C:\PolySpace\polyspace_project\Verification_1\Result_ | 9 AH-SRUNS  | completed | 15-Dec-2010, 16:03:33 | С         |
| ··· <mark>24</mark>                       | username   | example_project | C:\PolySpace\polyspace_project\Verification_1\Result_ | 10 AH-SRUNS | running   | 15-Dec-2010, 16:03:33 | С         |
| 25  | username   | example_project | C:\PolySpace\polyspace_project\Verification_1\Result_ | 11          | queued    | 15-Dec-2010, 16:03:21 | С         |
|   |            |                 |   |             |           |                       |           |
|   |            |                 |   |             |           |                       |           |
|   |            |                 |   |             |           |                       |           |
| onnec                                     | ted to Que | ue Manager AH-3 | SRUNSTRO.dhcp.mathworks.com                           |             |           | l                     | Jser mode |

**Tip** You can also open the Polyspace Queue Manager Interface by clicking the Polyspace Queue Manager icon an on the Run-Time Checks perspective toolbar.

- **2** Point anywhere in the row for ID 1.
- **3** Right-click to open the context menu for this verification.

| Follo | w Progress                            |
|-------|---------------------------------------|
| View  | Log File                              |
| Dow   | nload Results                         |
| Dow   | nload Results in New Polyspace Window |
| Dow   | nload Results and Remove From Queue   |
| Move  | e Down in Queue                       |
| Stop. |                                       |
| Stop  | and Download Results                  |
| Stop  | and Remove From Queue                 |
|       |                                       |

4 Select View log file.

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

```
📴 View Log File
                                                                  x
Generating results in a spreadsheet format in C:\PolySpace\PolySpace
Generation complete
***
*** Software Safety Analysis Level 4 done
***
                   Ending at: Dec 15, 2010 16:4:26
User time for pass4: 00:00:02.34 (2.3real, 2.3u + 0s)
Generating remote file
Done
User time for polyspace-c: 00:00:49.27 (49.3real, 49.3u + 0s (0.1gc))
***
                                                                    Ξ
*** End of Polyspace Verifier analysis
                                                                     ÷
د بد بد
٠.
              Ш
                                 Close
```

- 5 Click Close to close the window.
- **6** Select **Follow Progress** from the context menu.

The Progress Monitor opens.

|   | <u>H</u> elp    |              |              |            |             |         |
|---|-----------------|--------------|--------------|------------|-------------|---------|
| Progress Monitor                                |                 |              |              |            |             | 84      |
| Intermediate: 100%                              | Level0: 100%    | Level1: 100% | Level2: 100% | Level3: 0% | Level4 : 0% | Total   |
| 00:00:03  | 00:00:21        | 00:00:14     | 00:00:03     | 00:00:01   | 00:00:00    | 00:00:4 |
|   |                 |              |              |            |             |         |
|   |                 |              |              |            |             |         |
|   |                 |              |              |            |             |         |
|   |                 |              |              |            |             |         |
| Nuclear Charles                                 |                 |              |              |            |             | 8 P     |
| Verification Statistics                         |                 |              |              |            |             | 0' 4    |
| C   |                 |              |              |            |             |         |
|   | 1 : 248         |              |              |            |             |         |
| Number of lines without co                      |                 |              |              |            |             |         |
| *********************                           | ******          |              |              |            |             |         |
| Automatically stubbed pure                      | functions :     |              |              |            |             |         |
| ****************************                    | *************   |              |              |            |             |         |
| andom_int<br>andom_float                        |                 |              |              |            |             |         |
| get_bus_status                                  |                 |              |              |            |             |         |
|   |                 |              |              |            |             |         |
| *********************                           | *************** |              |              |            |             |         |
| Stats on aliases :                              | *************   |              |              |            |             |         |
| Some stats on aliases comp                      | utation:        |              |              |            |             |         |
| Number of invisibles:                           | 1               |              |              |            |             |         |
|   | 0               |              |              |            |             |         |
| Number of alias reads:<br>Number of pma writes: | 5               |              |              |            |             |         |

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. Follow the next steps to view the logs:

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

**Note** You can search the logs. In the **Search in the log** box, enter a search term and click the left arrows to search backward or the right arrows to search forward.

- 7 Select File > Quit to close the progress window.
- 8 Wait for the verification to finish.

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

| •                    | ions Help |                 | Dearth China  | 0011     | Charles   | Dete                  | 1        |
|----------------------|-----------|-----------------|---|----------|-----------|-----------------------|----------|
| ID                   | Author    | Application     | Results folder  | CPU      | Status    | Date                  | Language |
| 23                   | username  | example_project | C:\PolySpace\polyspace_project\Verification_1\Result_9  | AH-SRUNS | completed | 15-Dec-2010, 16:03:33 | С        |
| ···· <mark>24</mark> | username  | example_project | C:\PolySpace\polyspace_project\Verification_1\Result_10 | AH-SRUNS | completed | 15-Dec-2010, 16:04:23 | С        |
| 25                   | username  | example_project | C:\PolySpace\polyspace_project\Verification_1\Result_11 | AH-SRUNS | completed | 15-Dec-2010, 16:05:13 | С        |
|                      |           |                 |   |          |           |                       |          |
|                      |           |                 |   |          |           |                       |          |
|                      |           |                 |   |          |           |                       |          |
|                      |           |                 |   |          |           |                       |          |
|                      |           |                 |   |          |           |                       | ser mod  |

### **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.

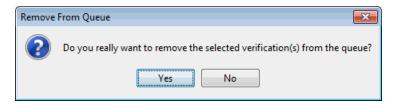
**Note** You can manually download verification results to another location on your client system, or to other client systems.

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 to confirm 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 Run-Time Checks perspective. You review results from the verification in Chapter 4, "Reviewing Verification Results".

## **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

If your computer does not have the minimum hardware requirements. the verification fails. For information about the hardware requirements, go to:

www.mathworks.com/products/polyspaceclientc/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.
- In the General section of the Analysis options, select the **Continue with current configuration option** and run the verification again.

#### You Did Not Specify the Location of Include 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 include files.

include.h: No such file or folder

For information on how to specify the location of include files, see "Creating a New Project to Verify the Example C File" on page 2-7.

#### **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.

| 🐣 Polyspace Pre  | ferences   |   |                                   | ×                  |  |  |  |  |  |  |
|--|--|---|-----------------------------------|--------------------|--|--|--|--|--|--|
| Tools Menu   | Review statuses  | Assistant configuration   | Miscellaneous                     | Character encoding |  |  |  |  |  |  |
| Serve  | r configuration  | Results folder  | Editors                           | Generic targets    |  |  |  |  |  |  |
| Remote configura   | Remote configuration   |   |                                   |                    |  |  |  |  |  |  |
| Note: Send to Polyspace server option is mandatory when the project contains multitasking options.<br>The multitasking options will be ignored otherwise.  |  |   |                                   |                    |  |  |  |  |  |  |
|  | Automatically detect the remote server     Automatically detect     Automaticaly detect     Automatically detect     Au |   |                                   |                    |  |  |  |  |  |  |
| O Use the follo  | ○ Use the following server and port : localhost           12427           12427  |   |                                   |                    |  |  |  |  |  |  |
| The server nar   | me "localhost" can be used i   | if the server is the local machine.   |                                   |                    |  |  |  |  |  |  |
| Metrics configura  | tion   |   |                                   |                    |  |  |  |  |  |  |
| Polyspace Metrics allows you, through a web browser, to drill down to specific coding rule violations and run-time checks. If you want to view or classify these items as defects within Polyspace, you click the item. Polyspace opens with the specific item displayed. However, this requires the downloading of result files from the Polyspace Metrics web interface to a locally accessible folder. On this tab, you specify how result files are downloaded from the Polyspace Metrics web interface and when justifications are saved in the Polyspace Metrics database.<br>If you select this check box, results are downloaded to the folder where the verification was launched. If this launch folder does not exist, results are downloaded to the location specified in the Folder field.<br>Otherwise, a file browser allows you to select the download location. |  |   |                                   |                    |  |  |  |  |  |  |
| Folder:  | sults automatically  | unstro\Documents\Polyspace Wor  |                                   | - 4-               |  |  |  |  |  |  |
| Folder:  | (\mmas-oo-an)si  | unsuo (pocuments (Polyspace_wor   | kspace pownloadedke               | suits              |  |  |  |  |  |  |
| If you select this check box, the save action (Ctrl+S) will save your justifications in the local results folder and the Polyspace<br>Metrics database. If you do not select this check box, the save action will save justifications in the local results folder only. If you<br>want to save justifications in the Polyspace Metrics database, click the 'Save in database' button.  |  |   |                                   |                    |  |  |  |  |  |  |
| Port used to com   | municate with the Polyspac   | e Metrics web interface.  |                                   |                    |  |  |  |  |  |  |
| Port number:   | 12428  |   |                                   |                    |  |  |  |  |  |  |
|  | etrics web interface URL is o<br>er can be configured above  | defined as follow: http:// <remotes< td=""><td>erver&gt;:<portnumber></portnumber></td><td></td></remotes<> | erver>: <portnumber></portnumber> |                    |  |  |  |  |  |  |
| Web server port  | number: 8080   |   |                                   |                    |  |  |  |  |  |  |
|  |  |   |                                   |                    |  |  |  |  |  |  |
|  |  |   | <u> <u>o</u>k</u>                 | Apply Cancel       |  |  |  |  |  |  |

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 the *Polyspace Installation Guide*.

# **Using Polyspace In One Click to Launch Verification**

#### In this section ...

"Overview of Polyspace In One Click" on page 3-23

"Setting the Active Project" on page 3-23

"Sending Files to Polyspace Software" on page 3-25

## **Overview of Polyspace In One Click**

In a Microsoft Windows environment, Polyspace software provides a convenient way to streamline your work when you want to verify several files using the same set of options. Once you have set up a project file that has the options that you want, you designate that project as the *active project*, and then send the source files to Polyspace software for verification. You do not have to update the project with source file information. This process is called *Polyspace In One Click*.

In this part of the tutorial, using Polyspace In One Click, you learn how to:

- 1 Set the active project.
- 2 Send files to Polyspace software for verification.

# **Setting the Active Project**

The active project is the project that Polyspace In One Click uses to verify the files that you select. Once you have set an active project, it remains active until you change the active project. Polyspace software uses the analysis options from the project; it does not use the source files or results folder from the project.

To set the active project:

**1** In the taskbar area of your Windows desktop, right-click the Polyspace In One Click icon:



The context menu appears.

|           | Set active project          | • |
|-----------|-----------------------------|---|
| <u>88</u> | Polyspace - Run-Time Checks |   |
| <b>1</b>  | Polyspace - Project Manager |   |
| <b>3</b>  | Spooler                     |   |
|           | Help                        | • |
|           | Exit                        |   |

2 Select Set active project > Browse.

The Please set an active project dialog box opens.

| Please set an active project.  |   | <b>—</b>   |
|--|---|--|
| 😋 🗢 🛡 🖉 « OSDisk (C:)  | )  PolySpace  polyspace_project   | Search polyspace_project   |
| Organize 🔻 New folder  |   | ## <b>-</b> 🗋 🔞  |
| Computer<br>CoDisk (C:)<br>Adoc_ja\$ (\jana:<br>Share\$ (\infnas-i<br>win32\$ (\infnas-i<br>win32\$ (\infnas-i<br>win32\$ (\infnas-i<br>bat\$ (\infnas-01<br>e hub (\\DFS-01-A<br>e qe\$ (\infnas-01<br>e srunstro (\math<br>srunstro (\math<br>winfnas-01-<br>e mathworks | <pre>includes includes sources Verification_1  sample_project.cfg</pre> |  |
| File <u>n</u> ar   | ne: example_project.cfg   | <ul> <li>Polyspace configuration files (* ▼</li> <li><u>Open</u> Cancel</li> </ul> |

- **3** Navigate to polyspace\_project.
- **4** Select example\_project.cfg.

**5** Click **Open** to apply the changes and close the dialog box.

## Sending Files to Polyspace Software

You can send several files to Polyspace software for verification. For this tutorial, you send one file, example.c.

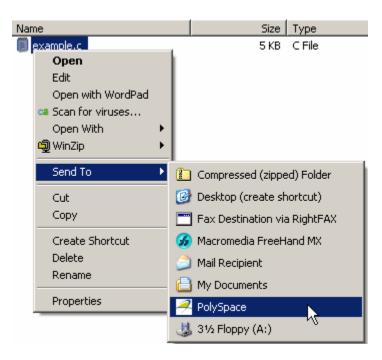
To send example.c to Polyspace software for verification:

- 1 Navigate to the folder polyspace\_project\sources.
- 2 Right-click the file example.c.

The context menu appears.

| Name   |                    |    |
|--------|--------------------|----|
| 📋 exar | pole c             | _  |
| _      | Open               |    |
|        | Edit               |    |
|        | Open with WordPad  |    |
|        | 端 Scan for viruses |    |
|        | Open With          | ъI |
|        | 🗐 WinZip           | •  |
|        | Send To            | •  |
|        | Cut                |    |
|        | Сору               |    |
|        | Create Shortcut    |    |
|        | Delete             |    |
|        | Rename             |    |
|        | Properties         |    |

**3** Select **Send To > Polyspace**.



The Polyspace basic settings dialog box appears.

| Polyspace basic settings [ C ]         |   |
|--|---|
| Settings                               |   |
| Precision                              | 02 💌                                    |
| Passes                                 | Pass2 (Software Safety Analysis level 💌 |
| Results folder                         | C:\PolySpace\polyspace_project          |
| Verification Mode Settings             |   |
| Variables written before loop          | None                                    |
| Functions called in loop               | AI                                      |
| Functions called before loop           |   |
| Scope                                  |   |
| C:\PolySpace\polyspace_project\sources | \example.c                              |
|  | -                                       |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
| Send to Polyspace S                    | Server 💽 Start 🔇 Cancel                 |

**4** Make sure that the **Results folder** is polyspace\_project.

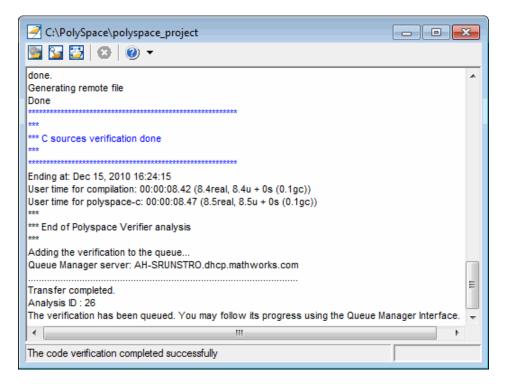
| Settings       |  |
|----------------|--|
| Precision      | 02                                       |
| Passes         | Pass2 (Software Safety Analysis level 2) |
| Results folder | C:\PolySpace\polyspace_project           |

5 If the Send to Polyspace Server option is not already selected, select it.

**6** Leave the default values for the other parameters.

Click Start.

The verification log opens.



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

• You see the following message in the log:

```
End of Polyspace Verifier analysis
```

• A message in the log states that the verification was transferred to the server and gives you the identification number (Analysis ID) for the verification. For this verification, the identification number is 1.

- Monitor the verification using the Spooler. For information on using the Spooler to monitor a verification on a server, see "Monitoring Progress Using Queue Manager" on page 3-14.
- When the verification is complete, download the results to polyspace\_project\results. For information on downloading results from a server to a client, see "Removing Verification Results from the Server" on page 3-19

You review the results in Chapter 4, "Reviewing Verification Results".

# Launching Client Verification from Project Manager

#### In this section ...

"Starting the Verification" on page 3-30

"Monitoring the Progress of the Verification" on page 3-32

"Completing Verification" on page 3-33

"Stopping the Verification Before It is Complete" on page 3-34

## **Starting the Verification**

For the best performance, run verifications on a server. If the server is busy or you want to verify a small file, 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 If the project example\_project.cfg is not already open, open the project.

For information about opening a project, see "Preparing for Verification" on page 3-4.

**2** Clear the **Send to Polyspace Server** check box in the General Analysis options.

| Name                                    | Value     | Internal name              |
|---|-----------|----------------------------|
| Analysis options                        |           |                            |
| General                                 |           |                            |
| Send to Polyspace Server                | <b>V</b>  | -server                    |
| Add to results repository               |           | -add-to-results-repository |
| ·····Keep all preliminary results files |           | -keep-all-files            |
| Calculate code metrics                  |           | -code-metrics              |
| -Report Generation                      |           |                            |
| Report template name                    | Developer | <br>-report-template       |
| Output format                           | RTF       | <br>-report-output-format  |
|   |           |                            |
| ⊕Compliance with standards              |           |                            |
|   |           |                            |
| Precision/Scaling                       |           |                            |
|   |           |                            |

- **3** If you see a warning that multitasking is not available when you run a verification on the client, click **OK** to continue and close the message box. This warning appears only when you clear the **Send to Polyspace Server** check box.
- **4** Click the **Run** button **Run** on the Project Manager toolbar.
- **5** If you see a caution 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 you see the message Verification process failed, click **OK** and go to "Troubleshooting a Failed Verification" on page 3-20.

## 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.

|   | 🖩 Progress Monitor - [C:\PolySpace\polyspace_project\Verification_1\Result_1] ローマン |  |              |             |             |  |
|---|--|--|--------------|-------------|-------------|--|
|   |  | Verification is running on server with ID: 1 |              |             |             |  |
|   | Intermediate: 100%   | Level0: 100%                                 | Level 1: 89% | Level2 : 0% | Level3 : 0% |  |
|   | 00:00:03   | 00:00:20                                     | 00:00:13     | 00:00:00    | 00:00:00    |  |
|   |  |  |              |             |             |  |
|   |  |  |              |             |             |  |
| 4 |  | m  |              |             | 4           |  |

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

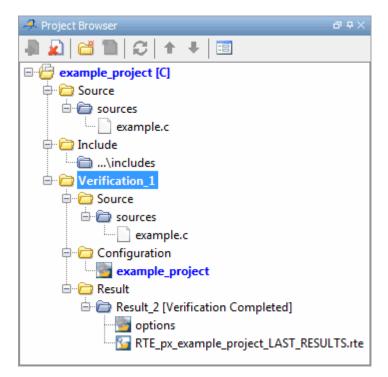
The logs report additional information about the progress of the verification. To view a log, click the button for that log. The information appears in the log display area at the bottom of the Project Manager window. Follow the next steps to view the logs:

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

**Note** You can search the logs. In the **Search in the log** box, enter a search term and click the left arrows to search backward or the right arrows to search forward.

## **Completing Verification**

When the verification is complete, the message "Verification Completed" appears at the bottom of the Project Manager window, and the results appear in the Project Browser.



In the tutorial Chapter 4, "Reviewing Verification Results", you open the Run-Time Checks perspective and review the verification results.

## Stopping the Verification Before It is Complete

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 over from the beginning.

To stop a verification:

1 Click the **Stop** button Stop on the Project Manager toolbar.

A warning dialog box opens.

| Polyspac | ce Verification Warning                           |
|----------|---|
|          | Do you really want to stop the current execution? |
|          | Yes No  |

2 Click Yes.

The verification stops and the message Verification process stopped appears.

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.

# 4

## Reviewing Verification Results

- "About Reviewing Verification Results Tutorial" on page 4-2
- "Opening Verification Results" on page 4-3
- "Exploring Run-Time Checks Perspective" on page 4-4
- "Reviewing Results in Manual Mode" on page 4-8
- "Reviewing Results in Assistant Mode" on page 4-23
- "Automatically Testing Unproven Code" on page 4-30
- "Generating Reports of Verification Results" on page 4-31

## **About Reviewing Verification Results Tutorial**

#### In this section...

"Overview" on page 4-2

"Before You Start" on page 4-2

## **Overview**

In the previous tutorial, Chapter 3, "Running a Verification", you completed a verification of example.c. In this tutorial, you explore the verification results.

The Polyspace verification environment contains a Run-Time Checks perspective that you use to review results. In this tutorial, you learn:

1 How to use the Run-Time Checks perspective, including how to:

- Open the Run-Time Checks perspective and view verification results.
- Explore results in expert mode.
- Explore results in assistant mode.
- Generate reports.
- **2** How to interpret the color-coding that Polyspace software uses to identify the severity of an error.
- **3** How to find the location of an error in the source code.

#### **Before You Start**

Before starting this tutorial, be sure to complete the tutorial Chapter 3, "Running a Verification".

In this tutorial, you use the verification results in this file:

```
polyspace_project\Verification_(1)\Result_(1)\
RTE_px_example_project_LAST_RESULTS.rte.
```

## **Opening Verification Results**

#### In this section ...

"Opening Run-Time Checks Perspective" on page 4-3

"Opening Verification Results" on page 4-3

## **Opening Run-Time Checks Perspective**

You use the Run-Time Checks perspective to review verification results.

To open the Run-Time Checks perspective:

• Select the **Run Time Checks** button Verification Environment toolbar.

## **Opening Verification Results**

To open the verification results:

1 Select File > Open Result.

The Please select a file dialog box opens.

2 Navigate to the results folder:

polyspace\_project\Verification\_(1)\Result\_(1).

- **3** Select the file RTE\_px\_example\_project\_LAST\_RESULTS.rte.
- 4 Click Open.

The results appear in the Run-Time Checks perspective.

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

## **Exploring Run-Time Checks Perspective**

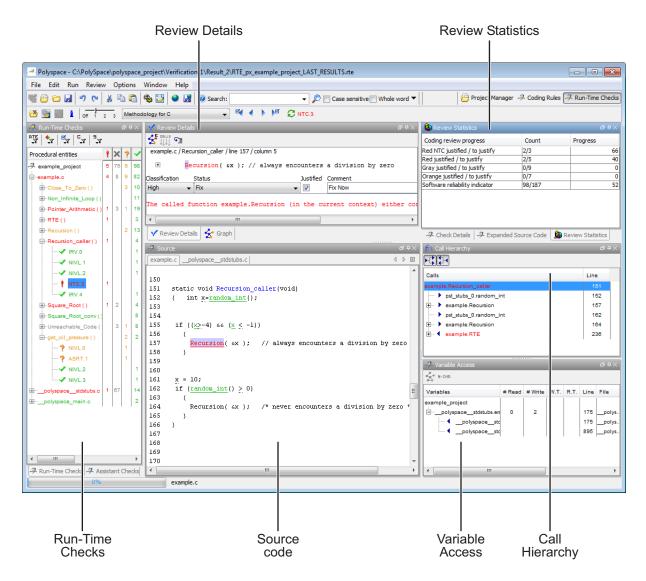
#### In this section...

"Overview" on page 4-4

"Reviewing the Run-Time Checks Pane" on page 4-6

## **Overview**

The Run-Time Checks perspective looks like the following figure.



The Run-Time Checks perspective has six sections below the toolbar. Each section provides a different view of the results. The following table describes these views.

| This Pane  | Displays  |
|--|---|
| Run-Time Checks<br>(Procedural entities view)      | List of the checks (diagnostics) for<br>each file and function in the project                               |
| Source<br>(Source code view)                       | Source code for a selected check in the procedural entities view  |
| Review Statistics<br>(Coding review progress view) | Statistics about the review progress<br>for checks with the same type and<br>category as the selected check |
| Review Details<br>(Selected check view)            | Details about the selected check  |
| Variable Access<br>(Variables view )               | Information about global variables declared in the source code  |
| Call Hierarchy<br>(Call tree view)                 | Tree structure of function calls  |

You can resize or hide any of these sections. You learn more about the Run-Time Checks perspective later in this tutorial.

## **Reviewing the Run-Time Checks Pane**

The Run-Time Checks pane displays a table with information about the diagnostics for each file in the project. The Run-Time Checks pane is also called the Procedural entities view

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

| 🚑 Run-Time Checks   |   |    |   |    |      |     |     | 9 th X              |
|---------------------|---|----|---|----|------|-----|-----|---------------------|
|                     |   |    |   |    |      |     |     |                     |
| Procedural entities | 1 | х  | ? | ~  | Line | Col | 8   | Details             |
| 🛹 example_project   | 5 | 38 | 7 | 14 |      |     | 89  |                     |
| example.c           | 4 | 5  | 7 | 14 | 1    |     | 77  | example.c           |
| polyspacestdstubs.c | 1 | 33 |   |    | 1    |     | 100 | polyspacestdstubs.c |
| polyspace_main.c    |   |    |   |    | 1    |     | 0   | polyspace_main.c    |

**4-6** 

The file example.c is red because it has a run-time error. Polyspace software assigns each file the color of the most severe error found in that file. The first column of the table in the Procedural Entities View is the procedural entity (the file or function). The following table describes some of the other columns in the procedural entities view.

| Column<br>Heading | Indicates   |
|-------------------|---|
| 1                 | 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)  |
| <b>&gt;</b>       | Number of green checks (operations where an error never occurs)   |
| *                 | Selectivity of the verification (percentage of checks that are<br>not orange)<br>This is an indication of the level of proof. |

**Note** You can select which columns appear in the procedural entities view by editing the preferences.

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

## **Reviewing Results in Manual Mode**

#### In this section ...

"What Is Manual Mode?" on page 4-8 "Switching to Manual Mode" on page 4-8 "Reviewing Checks in Manual Mode" on page 4-8 "Reviewing Additional Examples of Checks" on page 4-14 "Filtering Checks" on page 4-19

#### What Is Manual Mode?

In manual mode, you see all checks in the Run-Time Checks perspective. You decide which checks to review and in what order to review them.

## Switching to Manual Mode

By default, the Run-Time Checks perspective opens in assistant mode. To switch from assistant to manual mode:

• Move the Assistant slider to **Off** in the Run-Time Checks toolbar.

| 0 | 1 | 2        | 3 |
|---|---|----------|---|
|   |   | <u> </u> |   |

The toolbar displays buttons and menus specific to expert mode.

#### **Reviewing Checks in Manual Mode**

In this part of the tutorial, you learn how to use the Run-Time Checks perspective to examine verification checks. This part of the tutorial covers:

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

#### Selecting a Check to Review

In the procedural entities view, example.c is red, indicating that this file has at least one red check. To review a red check in example.c:

- 1 In the procedural entities section of the Run-Time Checks pane, expand example.c.
- 2 Expand the red procedure Pointer\_Arithmetic().

| -Pointer_  | Arithmetic ( ) | 1 | 2 | 1 | 6 | 89  | 12 | 90 | example.c                             |
|------------|----------------|---|---|---|---|-----|----|----|---------------------------------------|
|            | OVFL.2         |   |   |   | 1 | 94  | 23 |    | Operation [+] on scalar does not      |
|            | IDP.3          |   |   |   | 1 | 96  | 6  |    | Pointer is within its bounds          |
|            | IRV.6          |   |   |   | 1 | 100 | 5  |    | Function returns an initialized v     |
|            | IRV.7          |   |   |   | 1 | 102 | 9  |    | Function returns an initialized v     |
| ···· • •   | IDP.8          | 1 |   |   |   | 104 | 10 |    | Error : pointer is outside its bounds |
| - <b>X</b> | UNR.10         |   | 1 |   |   | 107 | 8  |    | Unreachable code                      |
| - <b>X</b> | OVFL.12        |   | 1 |   |   | 108 | 11 |    | Unreachable check : overflow o        |
|            | IRV.13         |   |   |   | 1 | 112 | 6  |    | Function returns an initialized v     |
| ~ ?        | IDP.15         |   |   | 1 |   | 114 | 16 |    | Warning : pointer may be outsid       |
|            | IDP.22         |   |   |   | 1 | 119 | 6  |    | Pointer is within its bounds          |

A color-coded list of the checks performed on Pointer\_Arithmetic() opens.

In the list of checks, each item has an acronym that identifies the type of check and a number. For example, in IDP.8, IDP stands for Illegal Dereferenced Pointer. For more information, see "Check Descriptions" in the *Polyspace Products for C Reference*.

3 Click the red IDP.8.

The Source pane displays the section of source code where this error occurs.

| A Source  |  | ٥ | Ψ×   |
|-----------|--|---|------|
| example.c | \  | Þ | Ξ    |
| 92        | int i, *p = array;                       |   | *    |
| 93        |  |   |      |
| 94        | for(i = 0; i < 100; i++)                 |   |      |
| 95        | {  |   |      |
| 96        | $\frac{*p}{2} = 0;$                      |   |      |
| 97        | <u>p</u> ++;                             |   |      |
| 98        | }  |   |      |
| 99        |  |   | Ξ    |
| 100       | $if(get_bus_status() \ge 0)$             |   |      |
| 101       | {  |   |      |
| 102       | <pre>if(get_oil_pressure() &gt; 0)</pre> |   |      |
| 103       | {  |   |      |
| 104       | $\frac{*p}{p}$ = 5; /* Out of bounds */  |   |      |
| 105       | }  |   |      |
| 106       | else                                     |   |      |
| 107       | <u>{</u>                                 |   |      |
| 108       | <u>i++</u> ;                             |   |      |
| 109       | }  |   |      |
| 110       | }  |   | Ŧ    |
| •         | III                                      |   | - Þ. |

**4** At line 104 of the code, click the red code.

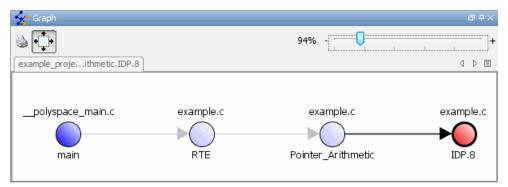
An error message box opens indicating that when the pointer **p** is dereferenced, it is outside of its bounds. At line 92, **p** points to the start of **array** which has 100 elements. The for loop starting at line 94 initializes the elements of **array** to 0. This for loop leaves **p** pointing to the location after the last element of **array**.

#### **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 IDP.8 check in Pointer\_Arithmetic():

- I Expand Pointer\_Arithmetic().
- 2 Click the red IDP.8.
- 3 Click the call graph button in the Review Details toolbar.

A window displays the call graph.



The code associated with IDP.9 is in Pointer\_Arithmetic. The generated main function calls RTE, which calls Pointer\_Arithmetic.

#### **Tracking Review Progress**

You can keep track of the checks that you have reviewed by marking them. To mark that you have reviewed the red IDP.8 check in Pointer\_Arithmetic():

- I Expand Pointer\_Arithmetic().
- 2 Click the red IDP.8.

The Review Statistics pane displays a table with statistics about the review progress for that category and severity of error.

| 🙆 Review Statistics            |                         |                   | 0  |
|--------------------------------|-------------------------|-------------------|----|
| Coding review progress         | Count                   | Progress          |    |
| Red IDP justified / to justify | 0/1                     |                   | 0  |
| Red justified / to justify     | 0/5                     |                   | 0  |
| Gray justified / to justify    | 0/75                    |                   | 0  |
| Orange justified / to justify  | 0/9                     |                   | 0  |
| Software reliability indicator | 98/187                  |                   | 52 |
|                                |                         |                   |    |
|                                |                         |                   |    |
| 🐥 Check Details 🛛 🐥 Expans     | ded Source Code 🛛 🥻 🕼 I | Review Statistics |    |

The **Count** column displays a ratio and the **Progress** column displays the equivalent percentage.

The first row displays the ratio of justified checks to total checks that have the same color and category as the current check. In this example, the first row displays the ratio of reviewed red IDP checks to total red IDP errors in the project.

The second row displays the ratio of justified checks to total checks that have the same color as the current check. In this example, this ratio is the ratio of red errors reviewed to total red errors in the project.

The last 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.

Information about the current check (the red IDP.8) appears in the Review Details window (Selected Check view).

| 💙 Review Details                                      |             | 리 무 X   |
|---|-------------|---|
| STATE STATE   |             |   |
| example.c / Pointer_Arithmetic / line 104 / column 10 |             |   |
| <pre> # *p = 5; /* Out of bounds */ </pre>            |             |   |
| Classification Status                                 | Justified   | Comment   |
| High 👻 Fix  | V           | Fix Now   |
|   | 400 in buff | 32, size: 32 bits):<br>Ter of 400 bytes, so is outside bounds<br>ble in: {Pointer_Arithmetic:array} |

- **3** After you review the check, select a **Classification** to describe the seriousness of the issue:
  - High
  - Medium
  - Low
  - Not a defect
- **4** Select a **Status** to describe how you intend to address the issue:
  - Fix
  - Improve
  - Investigate
  - Justify with annotations
  - No Action Planned
  - Other
  - Restart with different options
  - Undecided

**Note** You can also define your own statuses. See "Defining Custom Status ".

- **5** In the comment box, enter additional information about the check.
- 6 Select the check box to indicate that you have justified this check.

The **Coding review progress** part of the window updates the ratios of errors reviewed to total errors.

| Coding review progress         | Count  | Progress |
|--------------------------------|--------|----------|
| Red IDP justified / to justify | 1/1    | 100      |
| Red justified / to justify     | 1/5    | 20       |
| Gray justified / to justify    | 0/75   | 0        |
| Orange justified / to justify  | 0/9    | 0        |
| Software reliability indicator | 98/187 | 52       |
|                                |        |          |

## **Reviewing Additional Examples of Checks**

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

- "Example: Unreachable Code" on page 4-15
- "Example: Arithmetic Error" on page 4-16
- "Example: A Function with No Errors" on page 4-17
- "Example: Division by Zero" on page 4-17

#### **Example: Unreachable Code**

Unreachable code is code that never executes. Polyspace software displays unreachable code in gray. In the following example, you look at an example of unreachable code.

#### 1 In **Procedural Entities**, click Unreachable\_Code().

The source code view displays the source code for this function.

| 🛹 Sou | irce  | a b | $\times$ |
|-------|---|-----|----------|
| examp | ole.c d   | ⊳   | Ξ        |
| 199   | static void <mark>Unreachable_Code</mark> (void)                            |     | *        |
| 200   | /* Here we demonstrate PolySpace Verifier's ability                         | to  |          |
| 201   | identify unreachable sections of code due to the                            |     |          |
| 202   | value constraints placed on the variables.                                  |     |          |
| 203   | */  |     |          |
| 204   | $\{ \text{ int } x = \underline{\text{random int}}();$                      |     |          |
| 205   | $int \underline{y} = \underline{random_int}();$                             |     |          |
| 206   |   |     |          |
| 207   | if $(\underline{x} \geq \underline{y})$                                     |     |          |
| 208   | {   |     |          |
| 209   | $\underline{\mathbf{x}} = \underline{\mathbf{x}} - \underline{\mathbf{x}};$ |     |          |
| 210   | if $(\underline{x} \leq 0)$   |     |          |
| 211   | <u>(</u>  |     |          |
| 212   | $x = \underline{x} + 1;$  |     |          |
| 213   | }   |     |          |
| 214   | }   |     | Ŧ        |
| •     | III   |     | Þ.       |

**2** Examine the source code.

At line 212, the code x = x + 1 is never reached because the condition x < 0 is always false.

#### **Example:** Arithmetic Error

In the following example, Polyspace software detects a memory corruption error:

1 In Procedural entities, expand the red Square\_Root() function.

The source code view displays the source code for this function.

```
🐥 Source
                                                                    ∂₽×
example.c
                                                                   4 🕨 🗉
                                                                        *
179
      static void Square_Root_conv (double alpha, float *beta_pt)
180
            /* Perform arithmetic conversion of alpha to beta */
181
      {
182
        *beta_pt = (float)((1.5 + cos(alpha))/5.0);
183
      }
184
185
      static void Square Root (void)
186
      {
187
        double alpha = random_float();
188
        float beta;
189
        float gamma;
                                                                        Ξ
190
191
        Square_Root_conv (alpha, &beta);
192
193
        gamma = (float)sqrt(beta - 0.75);
                                              /* always sqrt(negative
194
      }
<.
                               ш
                                                                        Þ
```

**2** Examine the source code.

Because beta is always less than 0.75, the argument to the sqrt() function at line 193 is always negative.

#### **Example: A Function with No Errors**

In the following example, Polyspace software verifies code with a large number of iterations, and determines that the loop terminates and a variable does not overflow:

1 In Procedural entities, click the green Non\_Infinite\_Loop() function.

The source code view displays the source code for this function.

| A Source   | 9 t   |
|--|-------|
| example.c  | 4 ▷ 🗉 |
| <pre>66 static int Non Infinite Loop (void) 67 { const int big = 1073741821 ; /* 2**30-3 */ 68 int x=0, y=0;</pre> | *     |
| 69<br>70 while (1)<br>71   |       |
| 72 {<br>73 if $(\underline{y} \ge \underline{big})$ { break;}<br>74 $\underline{x} = \underline{x} \pm 2$ ;        |       |
| 75 $\mathbf{y} = \mathbf{x} / 2;$<br>76 }  |       |
| 77<br>78 $y = x / 100;$<br>79 return w   |       |
| 79 return <u>y</u> ;<br>80 }   | -     |
| <  | •     |

2 Examine the source code. The variable x never overflows because the while loop at line 70 terminates before x can overflow.

#### **Example:** Division by Zero

In the following example, Polyspace software detects division by zero:

1 In Procedural entities, expand Recursion().

The source code view displays the source code for this function.

| A Source 🛛 🕫 🕂 🗙   |
|--|
| example.c 4 b 🗉  |
| 137 static void Recursion (int* depth)                                       |
| 138 /* if depth<0, recursion will lead to division by zero */                |
| 139 { float advance;   |
| 140  |
| 141 *depth = *depth + 1;   |
| 142 advance = 1.0f/(float)(*depth); /* potential division by zero */         |
| 143  |
| 144  |
| 145 if (*depth < 50)   |
| 146 {  |
| 147 Recursion(depth);  |
| 148 }  |
| 149 }  |
| 150  |
| 151 static void Recursion_caller(void)                                       |
| <pre>152 { int x=random_int();</pre>   |
| 153  |
| 154  |
| 155 if $((\underline{x} \ge -4) \&\& (\underline{x} \le -1))$                |
| 156 {  |
| 157 <u>Recursion</u> ( $\alpha$ x ); // always encounters a division by zero |
| 158 }  |
| 159  |
| 160<br>161 	 x = 10;   |
| $\begin{array}{llllllllllllllllllllllllllllllllllll$                         |
| 163 {  |
| 165 (<br>164 Recursion( «x ); /* never encounters a division by zero */      |
| 165 }  |
|  |

2 Examine the Recursion() function.

When Recursion() is called with depth less than zero, the code at line 142 results in division by zero. The orange color indicates that this operation is a potential error (depending on the value of depth).

**3** Examine the red Recursion\_caller function.

The first call to Recursion() at line 157 is red because it calls Recursion() with depth less than zero, causing a division by zero. The second call to Recursion() at line 164 does not cause division by zero because it calls Recursion() with depth greater than zero.

## **Filtering Checks**

You can filter the checks that you see in the Run-Time Checks perspective so that you can 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, select one of the filter buttons in the Run-Time checks toolbar.



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

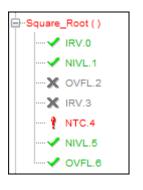
#### **Example: Filtering IRV Checks**

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

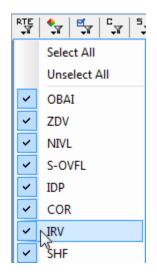
To filter IRV checks:

1 Expand Square\_Root().

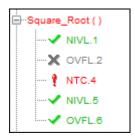
 ${\tt Square\_Root}$  ( ) has seven checks: four are green, one is red, and two are gray.



- 2 Click the RTE filter icon
- ${\bf 3}$  Clear the  ${\bf IRV}$  option.



The software hides the IRV check for  ${\tt Square\_Root}$  ( ).



**4** Select the IRV option to redisplay the IRV check.

**Note** When you filter a check category, red checks of that category are not hidden. For example, if you filter IDP checks, you still see IDP.8 under Pointer\_Arithmetic().

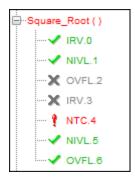
#### **Example: Filtering Green Checks**

You can use a Color filter to hide certain color checks. When a filter is enabled, you do not see that color check.

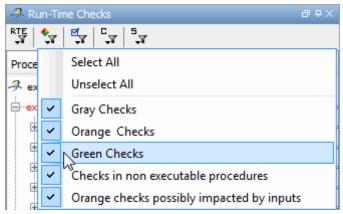
To filter green checks:

1 Expand Square\_Root().

 ${\tt Square\_Root}$  ( ) has seven checks: four are green, one is red, and two are gray.



- 2 Click the Color filter icon 🔄
- **3** Clear the **Green Checks** option.



The software hides the green checks.

| Square   | Root ( ) |
|----------|----------|
| ×        | OVFL.2   |
|          | IRV.3    |
| <b>१</b> | NTC.4    |

## **Reviewing Results in Assistant Mode**

#### In this section ...

"What Is Assistant Mode?" on page 4-23 "Switching to Assistant Mode" on page 4-23 "Selecting the Methodology and Criterion Level" on page 4-24 "Exploring Methodology for C" on page 4-24 "Reviewing Checks" on page 4-26 "Defining a Custom Methodology" on page 4-28

## What Is Assistant Mode?

By default, the Run-Time Checks perspective opens in assistant mode. In assistant mode, Polyspace software chooses the checks for you to review and the order in which you review them. Polyspace software presents checks in this order:

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

You learn about methodologies and criterion levels in "Selecting the Methodology and Criterion Level" on page 4-24.

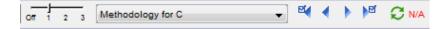
## Switching to Assistant Mode

To switch from expert to assistant mode:

• Move the Assistant slider to 1 in the Run-Time Checks toolbar.

|    |   |   | _ |
|----|---|---|---|
| or | 1 | 2 | 3 |
|    |   |   |   |

The Assistant Checks tab opens, displaying the checks you need to review, and the toolbar displays controls specific to assistant mode.



The controls for assistant mode include:

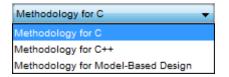
- A menu for selecting the review methodology for orange checks.
- A slider for selecting the criterion level within that methodology.
- Arrows for navigating through the reviews.

#### Selecting the Methodology and Criterion Level

A methodology defines which orange checks you review in assistant mode. Each methodology has three criterion levels, corresponding to different development phases, with increasing review requirements. As the criterion level increases, you review more checks.

To select the methodology and level for this tutorial:

1 Select Methodology for C from the methodology menu.



**2** If the level slider is not already at 1, move the slider to level 1.



#### **Exploring Methodology for C**

In this part of the tutorial, you examine **Methodology for C**, which defines the number of orange checks you review in assistant mode.

To examine the configuration for Methodology for C:

**1** Select **Options > Preferences**.

The Polyspace Preferences dialog box opens.

#### 2 Select the Assistant configuration tab.

The configuration for Methodology for C opens.

On the right side of the dialog box, a table shows the number of orange checks that you review for a given criterion and check category.

| Number of checks to review |             |             |             |
|----------------------------|-------------|-------------|-------------|
|                            | Criterion 1 | Criterion 2 | Criterion 3 |
| Common                     |             |             |             |
| ZDV                        | 5           | 20          | ALL         |
| NIVL                       | 10          | 50          | ALL         |
| S-OVFL                     | 10          | 50          | ALL         |
| COR                        |             | 10          | 10          |
| NIV                        |             | 0           | 10          |
| F-OVFL                     | 5           | 10          | 20          |
| ASRT                       |             | 5           | 20          |
| C & C++ only               |             |             |             |
| OBAI                       | 10          | 20          | ALL         |
| SHF                        | 5           | 10          | ALL         |
| IDP                        |             | 10          | 20          |
| NIP                        |             | 10          | 20          |
| C only                     |             |             |             |
| IRV                        | 5           | 20          | ALL         |

For example, the table specifies that you review five orange ZDV checks when you select criterion 1. The number of checks increases as you move from criterion 1 to criterion 3, reflecting the changing review requirements as you move through the development process. In the lower-left part of the dialog box, the section **Review threshold criterion** contains text that appears in the tooltip for the criterion slider on the Run-Time Checks toolbar.

| Configuration set   |             |
|---|-------------|
| Methodology for C   | •           |
| Set number of checks to review as percentage of green and justified orange checks |             |
| Review threshold criterion  |             |
| Criterion 1   | Fresh code  |
| Criterion 2   | Unit tested |
| Criterion 3   | Code review |

For the configuration Methodology for C, the following table lists the criterion names.

| Criterion | Name in the Tooltip |
|-----------|---------------------|
| 1         | Fresh code          |
| 2         | Unit tested         |
| 3         | Code Review         |

These names correspond to phases of the development process.

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

## **Reviewing Checks**

In assistant mode, you review checks in the order in which Polyspace software presents them:

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

Earlier in this tutorial, you selected Methodology for C, criterion l. In this part of the tutorial, you review the checks for example.c using this methodology and criterion. To navigate through these checks:

1 Click the forward arrow



The Assistant Checks tab shows the current check (IDP.8).

| Assistant Checks  | ₽₽× |
|---|-----|
| ÷ ↑   |     |
| Check   |     |
| example.c.Pointer_Arithmetic.IDP.8  |     |
| polyspacestdstubs.sqrt.ASRT.0   |     |
| example.c.Square_Root.NTC.4   |     |
| example.c.Recursion_caller.NTC.3  |     |
| <pre>example.c.Square_Root.NTC.4 example.c.Recursion_caller.NTC.3 example.c.RTE.NTC.3</pre>   |     |
| <pre>xpolyspacestdstubs.sqrt.NIVL.2</pre>   |     |
| <pre>xstdstubs.sqrt.UNR.3</pre>   |     |
| Xpolyspacestdstubs.sqrt.UNR.7   |     |
| Xpolyspacestdstubs.sqrt.UNR.9   |     |
| polyspace_stdstubs.sqrt.UNR.64  |     |
| polyspace_stdstubs.sqrt.UNR.66  |     |
| × example.c.Square_Root.OVFL.2  |     |
| × example.c.Pointer_Arithmetic.UNR.10   |     |
| <pre>x example.c.Unreachable_Code.UNR.8</pre>   |     |
|   |     |
| <pre>? example.c.get_oil_pressure.NIVL.0</pre>  |     |
| <ul> <li>example.c.Recursion.ZDV.7</li> <li>example.c.get_oil_pressure.NIVL.0</li> <li>example.c.Unreachable_Code.OVFL.5</li> <li>example.c.Close_To_Zero.OVFL.3</li> <li>example.c.Close_To_Zero.OVFL.6</li> </ul> |     |
| example.c.Close_To_Zero.OVFL.3  |     |
| example.c.Close_To_Zero.OVFL.6  |     |
| example.c.Close_To_Zero.OVFL.8  |     |
| example.c.Recursion.OVFL.6  |     |
|   |     |
| Assistant Checks  | s   |
| 0%  |     |
|   |     |

The source code view (lower right) displays the source for this check and the current check view (upper right) displays information about this check.

**Note** You can display the calling sequence and track review progress, as described in "Reviewing Results in Manual Mode" on page 4-8.

**2** Continue to click the forward arrow until you have gone through all of the checks.

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

| Wrappin | ig search  |
|---------|--|
| ?       | End of the set of checks under review.<br>Do you want to start again from the first check? |
|         | Yes No   |

3 Click No.

#### **Defining a Custom Methodology**

You cannot change the predefined methodologies, such as Methodology for C, but you can define your own methodology. In this part of the tutorial, you learn how to create and use your own methodology.

The methodology that you create is the Methodology for C with one change.

To define your custom methodology:

1 Select **Options > Preferences**.

The Polyspace Preferences dialog box opens.

- 2 Select the Assistant configuration tab.
- **3** Select Add a set from the Configuration set menu.

- **4** In the Create a new set dialog box, enter My methodology for the name and click **Enter** to close the dialog box.
- **5** Under the **Criterion 1** column, enter the number 1 next to **IDP**. Polyspace software selects up to one orange IDP for review.
- 6 Click OK to save the methodology and close the dialog box.

To use My methodology:

- **1** Select My methodology from the methodology menu.
- **2** If the level slider is not already at 1, move the slider to level 1.
- **3** Click the forward arrow **b** to review the checks.

With this methodology at criterion 1, you review the orange IDP.17 (you did not review IDP.17 earlier in the tutorial because the number of orange IDP checks in Methodology for C, criterion 1 is zero).

## **Automatically Testing Unproven Code**

Reviewing orange code to find true errors is a time-consuming task. You can use the Automatic Orange Tester (AOT) to automatically create and run test cases to identify errors in the orange code. The workflow for using the AOT is:

- 1 Set an option to indicate that you want to prepare automatic tests.
- 2 Run the verification to prepare the tests and verify the source code.
- **3** When the verification is finished, run the test cases.
- **4** Review the results.

To learn how to use the AOT, see "Automatically Testing Orange Code" in the *Polyspace Products for C User's Guide.* 

## **Generating Reports of Verification Results**

#### In this section...

"Polyspace Report Generator Overview" on page 4-31

"Generating Report for example.c" on page 4-32

## **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. Detailed results are sorted by type of check (Proven Run-Time Violations, Proven Unreachable Code Branches, Unreachable Functions, and Unproven Run-Time Checks).
- **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 comprehensive information on software quality objectives (SQO), including code metrics, code analysis (coding-rules checker results), code verification (run-time checks), and the configuration settings for the verification. The code

metrics section provides is the same information displayed in the Polyspace Metrics web interface.

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

- HTML
- PDF
- RTF
- Microsoft Word
- XML

**Note** Microsoft Word format is not available on UNIX platforms. If you select Word format on a UNIX platform, the software uses RTF format instead.

#### Generating Report for example.c

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

To generate a verification report:

1 If your verification results are not already open, open them.

#### 2 Select Run > Run Report > Run Report.

The Run Report dialog box opens.

| Run Report       |  |
|------------------|--|
| Select Report Te | mplate   |
| C: PolySpace     | PolySpace_Common\ReportGenerator\templates\CodingRules.rpt               |
|                  | PolySpace_Common\ReportGenerator\templates\Developer.rpt                 |
|                  | PolySpace_Common\ReportGenerator\templates\DeveloperReview.rpt           |
|                  | PolySpace_Common\ReportGenerator\templates\Developer_WithGreenChecks.rpt |
|                  | PolySpace_Common\ReportGenerator\templates\Quality.rpt                   |
| C: PolySpace     | PolySpace_Common\ReportGenerator\templates\SoftwareQualityObjectives.rpt |
|                  |  |
|                  |  |
|                  | <u>B</u> rowse   |
| Select Report Fo | rmat   |
| Output folder    | C:\PolySpace\polyspace_project\Verification_1\Result_2\Polyspace-Doc     |
| Output format    | RTF  |
|                  | Run Report         Cancel  |

- **3** In the Select Report Template section, select **Developer.rpt**.
- **4** In the Output folder section, select the \polyspace\_project folder.
- **5** Select PDF Output format.
- 6 Click Run Report.

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

# 5

## Checking MISRA C Compliance

- "About Checking MISRA C Compliance Tutorial" on page 5-2
- "Setting Up MISRA C Checking" on page 5-3
- "Running a Verification with MISRA C Checking" on page 5-14

## About Checking MISRA C Compliance Tutorial

#### In this section...

"Overview" on page 5-2

"Before You Start" on page 5-2

#### **Overview**

Polyspace software allows you to analyze code to demonstrate compliance with MISRA C 2004 standards.  $^{3}$ 

Applying coding rules can both reduce the number of orange checks in your verification results, and improve the quality of your code. Coding rules are the most efficient way to reduce orange checks.

To check compliance with coding rules, you set an option in your project and then run a verification. Polyspace software finds the violations during the compile phase of a verification. When you have addressed all coding rule violations, you run the verification again.

In this tutorial, you learn how to:

- 1 Create a second verification within your project.
- 2 Set an option for checking MISRA C compliance.
- **3** Select MISRA C rules to check.
- 4 Run a verification with MISRA C checking.
- 5 View coding rules violations using the Coding Rules perspective.

#### **Before You Start**

For this tutorial, you check the MISRA C compliance of the file example.c, using the project that you created in Chapter 2, "Setting Up a Polyspace Project".

<sup>3.</sup> MISRA and MISRA C are registered trademarks of MISRA Ltd., held on behalf of the MISRA Consortium.

# Setting Up MISRA C Checking

#### In this section ...

"Opening Your Example Project" on page 5-3 "Creating New Verification" on page 5-4 "Setting MISRA C Checking Option" on page 5-7 "Creating a MISRA C Rules File" on page 5-8 "Excluding Files from the MISRA C Checking" on page 5-11 "Configuring Text and XML Editors" on page 5-12 "Saving the Project" on page 5-13

## **Opening Your Example Project**

For this tutorial, you modify the project in example.cfg to include MISRA C checking. You use the Project Manager perspective to modify the project.

To open example\_project.cfg:

1 Select File > Open Project.

The Open a Polyspace project file dialog box opens.

- 2 Navigate to polyspace\_project.
- **3** Select example\_project.cfg.
- 4 Click **Open** to open the file and close the dialog box.

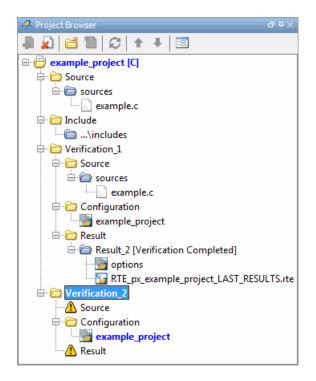
## **Creating New Verification**

A Polyspace project can contain multiple verifications. Each verification may verify a different set of source files, and may use different analysis options. In this section, you create a second verification to check coding rules compliance for the example.c file.

To create a new verification in example\_project:

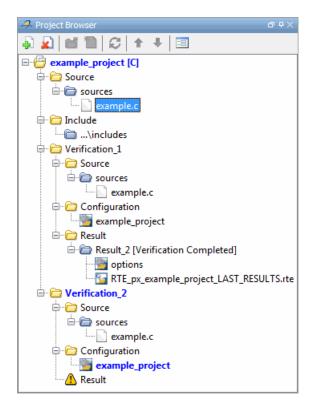
- 1 In the Project Browser, select example\_project [C].
- 2 Click the **Create a new verification** icon iii in the Project Browser toolbar.

A new verification, Verification\_(2), appears in the Project Browser.



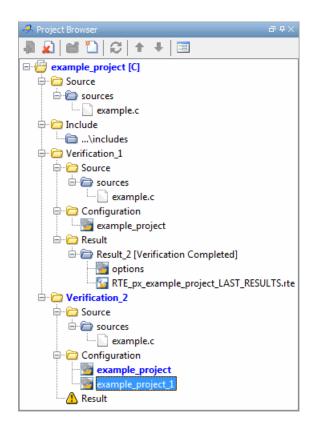
3 In the Project Browser Source tree, right-click example.c, and select Copy Source File to > Verification\_(2).

The example.c file appears in the Source tree of Verification\_(2).



- **4** Right-click the Configuration folder in Verification\_(2), and select **Create New Configuration**.
- 5 Right-click the example\_project\_1 configuration, and select Set as Active Configuration.

The Project Browser now looks like the following figure.



## **Setting MISRA C Checking Option**

You set up MISRA C checking by setting an analysis option and then selecting the rules to check. To set the MISRA C checking option:

- 1 Select the example\_project\_(1) Configuration in the Project Browser.
- 2 In the Analysis options part of the Configuration pane, expand the **Compliance with standards** option.
- 3 Select the Check MISRA C rules check box.
- 4 Expand the Check MISRA C rules option.

Two options appear, MISRA C rules configuration and Files and folders to ignore.

| Check MISRA C rules             | <b>V</b> |                         |
|---------------------------------|----------|-------------------------|
| ····MISRA C rules configuration | custom 👻 | <br>-misra2             |
| Files and folders to ignore     |          | <br>-includes-to-ignore |

These options allow you to specify which MISRA C rules to check and which, if any, files to exclude from the code analysis.

**5** In the MISRA C rules configuration drop-down list, select **custom**.

## **Creating a MISRA C Rules File**

You must have a rules file to run a verification with MISRA C checking. You can use an existing file or create a new one. You create a new rules file for this tutorial by:

- "Opening a New Rules File" on page 5-8
- "Setting All the Rules to Off" on page 5-9
- "Selecting the Rules to Check" on page 5-9

#### **Opening a New Rules File**

To open a new rules file:

1 Click the button with the right of the **Rules configuration** option.

A window for opening or creating a MISRA C rules file opens.

**2** Select File > New File.

A table of rules appears. For each rule, specify one of the following states.

| State   | Causes the verification to  |
|---------|---|
| Error   | End after the compile phase when this rule is violated.                             |
| Warning | Display warning message and<br>continue verification when this rule<br>is violated. |
| Off     | Skip checking of this rule.   |

**Note** The default state for most rules is Warning. The state for rules that have not yet been implemented is Off. Some rules always have a state of Error (you cannot change this state).

#### Setting All the Rules to Off

In this tutorial, you check only a few rules. Therefore, first set the state of all rules to Off. Later, you can select the specific rules that you want to check.

To set the state of all rules to Off:

- 1 From the Set the following state to all MISRA rules menu, select Off.
- 2 Click Go.

#### Selecting the Rules to Check

To select the rules to check for this tutorial:

- **1** Expand the set of rules named 16 Functions.
- 2 Select the Error column for 16.3.
- **3** Expand the set of rules named 17 Pointers and Arrays.
- 4 Select the Warning column for 17.4.

The completed rules table looks like the following figure:

| File   |           |            |        |
|--|-----------|------------|--------|
|  |           |            |        |
| Set the following state to all MI  | SRA C rul | es : Error | ▼ GC   |
| Rules  | Error     | Warning    | Off    |
| /ISRA C rules  |           |            |        |
| Number of rules by mode :  | 1         | 1          | 140    |
|  |           |            |        |
|  |           |            |        |
| 3 Documentation  |           |            |        |
| - 4 Character sets   |           |            |        |
| ∎5 Identifiers   |           |            |        |
| Erector 6 Types  |           |            |        |
| -7 Constants   |           |            |        |
| ∎~8 Declarations and definitions   |           |            |        |
| ••9 Initialisation   |           |            |        |
| ⊡…10 Arithmetic type conversions   |           |            |        |
| ⊡…11 Pointer type conversions  |           |            |        |
| ∎ 12 Expressions   |           |            |        |
| ⊡ 13 Control statement expressions   |           |            |        |
| ∎ 14 Control flow  |           |            |        |
|  |           |            |        |
| ∃16 Functions  | _         | -          | -      |
|  |           |            | 0      |
| 16.2 Functions shall not call themselves, either directly or indirectly.   | <u> </u>  |            | 0      |
| — 16.3 Identifiers shall be given for all of the parameters in a function prototy<br>— 16.4 The identifiers used in the declaration and definition of a function shall | 0         |            |        |
|  | 0         |            |        |
|  |           |            |        |
|  |           |            |        |
|  |           |            |        |
|  |           |            |        |
| 16.10 If a function returns error information, then that error information sh  | õ         | 0          |        |
| =-17 Pointer and arrays  |           |            |        |
| 17.1 Pointer arithmetic shall only be applied to pointers that address an arra   | 0         | 0          | 0      |
| 17.2 Pointer subtraction shall only be applied to pointers that address eleme  | ŏ         | - O        | 0      |
|  | 0         | 0          | 0      |
|  | 0         | ۲          | 0      |
|  | 0         | 0          | ۲      |
|  | 0         | 0          | ۲      |
|  |           |            |        |
| •  |           |            |        |
|  |           | ĸ          | Cancel |

5 Click OK to save the rules and close the window.

The Save as dialog box opens.

- 6 In File, enter misrac.txt
- 7 Click OK to save the file and close the dialog box.

## **Excluding Files from the MISRA C Checking**

You can exclude files from MISRA C checking. You might want to exclude some included files. To exclude math.h from the MISRA C checking of the project example.cfg:

1 Click the button is to the right of the Files and folders to ignore option.

The Files and folders to ignore dialog box opens.

|   | ×       |
|---|---------|
| Files and folders to ignore [-includes-to-ignore] |         |
|   |         |
| C: \PolySpace \polyspace_project \indudes         |         |
|   |         |
|   |         |
|   |         |
|   |         |
|   |         |
| ОК  | Cancel  |
|   | Caricer |
|   |         |

- **2** Click the folder icon
- **3** Navigate to the folder polyspace\_project\includes.
- 4 Select the file math.h.
- 5 Click OK.

The file math.h appears in the list of files to ignore.

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

## **Configuring Text and XML Editors**

Before you check MISRA<sup>®</sup> rules, configure your text and XML editors in the Preferences. Configuring text and XML editors allows you to view source files and MISRA reports directly from the Coding Rules perspective.

To configure your text and .XML editors:

#### 1 Select **Options > Preferences**.

The Preferences dialog box opens.

**2** Select the **Editors** tab.

| 🐣 Polyspace Pre   | ferences                               |                                       |               | ×                  |
|-------------------|--|---------------------------------------|---------------|--------------------|
| Tools Menu        | Review statuses                        | Assistant configuration               | Miscellaneous | Character encoding |
| Serve             | r configuration                        | Results folder                        | Editors       | Generic targets    |
| XML editor config | uration<br>path to a XML editor or use | the browse button.                    |               | E                  |
| XML Editor:       | C:\Program Files (x86)≬                | Aicrosoft Office \Office 12 \EXCEL.EX | E 🔁           |                    |
| Text Editor:      | bath to a text editor or use           | ws NT\Accessories\wordpad.exe         |               |                    |
| Arguments:        | Wordpad 🔹 \$FIL                        |                                       |               |                    |
| •                 |  |                                       |               |                    |
|                   |  |                                       | ОК            | Apply Cancel       |

**3** Specify an XML editor to use to view MISRA-C reports. For example:

C:\Program Files\MSOffice\Office12\EXCEL.EXE

**4** Specify a Text editor to use to view source files from the Project Manager logs. For example:

C:\Program Files\Windows NT\Accessories\wordpad.exe

- **5** Select your text editor in the Arguments drop-down menu to automatically specify the command line arguments for that editor.
  - Emacs
  - Notepad++
  - UltraEdit
  - VisualStudio
  - Wordpad

If you are using another text editor, select Custom from the drop-down menu, and specify the command line arguments for the text editor.

6 Click OK.

## **Saving the Project**

Save your project to save your new verification and analysis settings.

# **Running a Verification with MISRA C Checking**

#### In this section ...

"Starting the Verification" on page 5-14

"Examining MISRA C Violations" on page 5-16

"Opening MISRA-C Report" on page 5-20

## **Starting the Verification**

When you run a verification with the MISRA C option selected, the verification checks most of the MISRA C rules during the compile phase.<sup>4</sup>

If there is a violation of a rule with state Error, the verification stops.

To start the verification:

1 Select Verification\_(2) in the Project Browser.

2 Click the **Run** button **Run** on the Project Manager toolbar.

The verification fails because of MISRA C violations. The message "Verification Failed" appears at the bottom of the Project Manager perspective, and the Output Summary indicates that the verification has detected MISRA errors.

<sup>4.</sup> MISRA and MISRA C are registered trademarks of MISRA Ltd., held on behalf of the MISRA Consortium.

| 🛃 Outp           | out Summary - [C:\PolySpace\polyspace_project\Verification_2\Result_3 |      |      |     |
|------------------|---|------|------|-----|
| Search:          | +   |      |      |     |
| Class            | Description   | File | Line | Col |
| 1                | example_project for C verification start at Dec 15, 2010 18:33:40     |      |      |     |
| 1                | Verifier has detected MISRA error(s) in the code.                     |      |      |     |
| 1                | Exiting because of previous error                                     |      |      |     |
| ,<br>Detail:     |   |      |      |     |
| Error:<br>Verifi | er has detected MISRA error(s) in the code.                           |      |      |     |

## **Examining MISRA C Violations**

To examine the MISRA C violations:

1 Double-click MISRA-C-report.xml in the Project Browser Result folder.

The Coding Rules perspective appears, displaying a list of MISRA C violations.

| Polyspace - C:\PolySpace\po    | lyspace_project\                      | Verification_2 | \Result | _5\Poly   | Space-Doc\MI       | SRA-C-repor | t.xml [   | - • <b>·</b>     |
|--------------------------------|---------------------------------------|----------------|---------|-----------|--------------------|-------------|-----------|------------------|
| <u>File Edit R</u> un Review O | ptions <u>W</u> indow                 | / <u>H</u> elp |         |           |                    |             |           |                  |
| 📲 🗁 🗁 🔙 🔊 ୯୦ 👗 🖣               | ù 🛍 🖗 🔁                               | 🕺 🛃 🧕          |         |           |                    |             |           | Coding Rules     |
| <b>ĕ</b>                       |                                       |                |         |           |                    |             |           |                  |
| 🔎 Assistant Coding Rules 🗇 🕂 🗙 | 🧈 MISRA C                             |                |         |           |                    |             |           | a +×             |
| + +                            | Filter                                |                | F       | lide just | ified violated rul | es          |           | i                |
| Rule File Line Col             | w/e Rule                              | File           | Line    | Col       | Classification     | Status      | Justified | Comment          |
|                                | error 16.3                            | include.h      | 33      | 28        |                    |             |           |                  |
|                                | warning 17.4                          | example.c      | 97      | 7         |                    |             |           |                  |
|                                | warning 17.4                          | example.c      | 114     | 21        |                    |             |           |                  |
|                                | warning 17.4                          | example.c      | 118     | 14        |                    |             |           |                  |
|                                |                                       |                |         |           |                    |             |           |                  |
| 2                              | · · · · · · · · · · · · · · · · · · · |                |         |           |                    |             |           |                  |
| 🧘 Rule details                 |                                       |                |         |           |                    |             |           | 9 <del>4</del> X |
| 0%                             |                                       | Ready          |         |           |                    |             |           |                  |

**2** Click any of the violations to see a description of the violated rule, the full path of the file in which the violation was found, and the source code containing the violation.

| 🔫 Poly       | 🕊 Polyspace - C:\PolySpace\polyspace_project\Verification_2\Result_5\PolySpace-Doc\MISRA-C-report.xml |            |                       |           |         |             |        |           |                |              |             |            |       |
|--------------|---|------------|-----------------------|-----------|---------|-------------|--------|-----------|----------------|--------------|-------------|------------|-------|
| <u>F</u> ile | <u>File Edit Run Review Options Window H</u> elp  |            |                       |           |         |             |        |           |                |              |             |            |       |
| ۳ 🕒          | 🎬 🗁 🗁 🚽 🤊 🐃 👗 🏝 🎬 🧠 🔛 🥥   |            |                       |           |         |             |        |           |                |              |             |            |       |
| <b>č</b>     |   |            |                       |           |         |             |        |           |                |              |             |            |       |
| 🔎 Assi       | stant Cod   | ling Rules | s∂₽×                  | 🧢 MISRA   | A C     |             |        |           |                |              |             |            | ∂₽×   |
| + +          |   |            |                       | Filter    |         |             | 📄 Hide | justified | violated rules |              |             |            | 1     |
| Rule         | File  | Line       | Col                   | w/e       | Rule    | File        | Line   | Col       | Classification | Status       | Justified   | Comment    |       |
|              |   |            |                       | error     | 16.3    | indude.h    | 33     |           |                |              |             |            |       |
|              |   |            |                       | warning   | 17.4    | example.c   | 97     | -         |                |              |             |            |       |
|              |   |            |                       | warning   | 17.4    | example.c   | 114    |           |                |              |             |            |       |
|              |   |            |                       | warning   | 17.4    | example.c   | 118    | 14        |                |              |             |            |       |
|              |   |            |                       |           |         |             |        |           |                |              |             |            |       |
| 🔑 Rule       | details   |            |                       |           |         |             |        |           |                |              |             |            | ∂₽×   |
| Rule:        | 16.3 (  | error)     | : Ident               | tifiers s | shall b | e given fo  | r all  | of the    | parameters :   | in a functio | on prototyp | e declarat | tion. |
| File:        | C:\Pol  | ySpace     | <pre>&gt;polysp</pre> | pace_proj | ject\ir | ncludes\inc | lude.h | line 3    | 33 (column 2)  | 8)           |             |            |       |
|              | Source code<br> extern void Exec_One_Cycle (int);   |            |                       |           |         |             |        |           |                |              |             |            |       |
|              |   | 0          | %                     |           | F       | Ready       |        |           |                |              |             |            |       |

The log reports a violation of rule 16.3. A function prototype declaration in include.h is missing an identifier.

**3** Right-click the row containing the violation of rule 16.3, and select **Open Source File**.

| error   | 16.3 |                    |                                    |  |  |  |  |
|---------|------|--------------------|------------------------------------|--|--|--|--|
| warning | 17.4 | <del>و</del>       | Open Source File                   |  |  |  |  |
| warning | 17.4 |                    | Add Pre-Justification to Clipboard |  |  |  |  |
| warning | 17.4 |                    | Open MISRA-C Report                |  |  |  |  |
|         |      | а <mark>е</mark> р | Configure Editor                   |  |  |  |  |

The include.h file opens in your text editor.

-

**Note** Before you can open source files, you must configure a text editor. See "Configuring Text and XML Editors" on page 5-12.

```
🔚 🤚 🦿 🗢 🖬 include.h - WordPad
                                                 - - X
1日 -
      Home
            View
                                                    æ
        Courier New
                   • 11 • A A
                                🖅 🖅 🗄 - 🏥 -
Paste
        B I U abe X2 X2 - A -
                                              Insert Editing
                                물 좀 좀 좀 줄
                  Font
                                   Paragraph
Clipboard
 #ifndef INCLUDE H
#define INCLUDE H
#define true 1
#define false 0
#define checksum 1
#define new move 0
#define previous_move 1
#define MAX SIZE 10
/* automatically stubbed functions */
extern void SEND MESSAGE(int status, const char *message);
extern int read bus status(void);
extern int error_on_bus(void);
extern int read on bus(void);
/* internal functions */
void initialise current data (void);
void compute_new_coordonates(void);
void sort calibration(void);
int polynomia (int input);
       random int (void);
int
float random_float(void);
extern void partial init(int *new alt);
extern void RTE (void);
extern void Exec One Cycle (int);
                        ....
                                                0
                                     100% 😑 🕐
                                                       (\pm)
```

4 Correct the MISRA violation and run the verification again.

The verification then is completed, and the results will be the same as those from the tutorial in Chapter 3, "Running a Verification".

## **Opening MISRA-C Report**

After you check MISRA rules, you can generate an XML report containing all the errors and warnings reported by the MISRA-C checker.

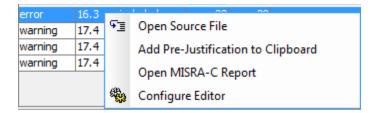
**Note** Before you can open a MISRA-C report, you must configure an XML editor. See "Configuring Text and XML Editors" on page 5-12.

To view the MISRA-C report:

**1** Click the **Coding Rules** button in the Polyspace Verification Environment toolbar.

A list of MISRA C violations appears in the Coding Rules perspective.

2 Right-click any row in the log, and select Open MISRA-C Report.



The report opens in your XML editor.

| Book2 - Microsoft Excel  |                      | Table Tools  | _ = X  |
|--|----------------------|--|--|
| Home Insert Page Layout Formulas Data Review View  | Add-Ins Acrobat      | Design   | @_ = ×   |
| Calibri ▼ 11 ▼ A Ă<br>Paste<br>Clipboard G<br>Font<br>Calibri ▼ 11 ▼ A Ă<br>E =<br>Paste<br>Clipboard G<br>Font<br>Clipboard G | General<br>S 7 % 7 % | Conditional Format Cell Formatting * as Table * Styles*     Styles | Σ ·<br>Sort & Find &<br>C · Filter · Select ·<br>Editing |
| Name 🔽 Mode 🔽 Report 🔽 File  | 🔹 Line 💌 Column 💌 M  | 1essage  | -  |
| 16.3 required error C:\PolySpace\polyspace_project\includes\include.   |                      | Identifiers shall be given for all of the parameter                |  |
| 17.4 required warning example.c  |                      | Array indexing shall be the only allowed form of                   |  |
| 17.4 required warning example.c  |                      | Array indexing shall be the only allowed form of                   |  |
| 17.4 required warning example.c  | 118 0                | Array indexing shall be the only allowed form of                   | pointer arithmetic.                                      |
|  |                      |  |  |
|  |                      |  |  |
|  |                      |  |  |
|  |                      |  |  |
|  |                      |  |  |
|  |                      |  |  |
|  |                      |  |  |
|  |                      |  |  |
|  |                      |  |  |
|  |                      |  |  |
|  |                      |  |  |
| If ← → H Sheet1 Sheet2 Sheet3 2  |                      |  |  |

# Index

## A

active project definition 3-23 setting 3-23 analysis options 2-13 MISRA C compliance 5-7 ANSI compliance 3-10 AOT. See Automatic Orange Tester assistant mode criterion 4-24 custom methodology 4-28 methodology 4-24 methodology for C 4-24 overview 4-23 reviewing checks 4-26 selection 4-23 use 4-23 4-26 Automatic Orange Tester overview 4-30

## С

call graph 4-11 call tree view 4-4 calling sequence 4-11 cfg. See configuration file client 1-5 3-2 installation 1-11 verification on 3-30 coding review progress view 4-4 4-11 Coding Rules perspective 1-5 color-coding of verification results 1-2 to 1-3 4-6 compile log Project Manager 3-13 3-32 Spooler 3-14 compile phase 3-10 compliance ANSI 3-10 MISRA C 1-3 5-1 configuration file

definition 2-3 custom methodology definition 4-28

#### D

default folder changing location 2-7 desktop file definition 2-3 division by zero example 4-17 downloading results 3-19 dsk. *See* desktop file

## E

expert mode filters 4-19

#### F

files includes 2-11 source 2-11 filters 4-19 folders includes 2-11 sources 2-11

#### Η

hardware requirements 3-20 help accessing 1-15

#### 

installation Polyspace Client for C/C++ 1-11 Polyspace products 1-11 Polyspace Server for C/C++ 1-11

## L

licenses obtaining 1-11 logs compile Project Manager 3-13 3-32 Spooler 3-14 full Project Manager 3-13 3-32 Spooler 3-14 stats Project Manager 3-13 3-32 Spooler 3-14 viewing Project Manager 3-13 3-32 Spooler 3-14

#### M

manual mode overview 4-8 selection 4-8 use 4-8 methodology for C 4-24 MISRA C compliance 1-3 analysis option 5-7 checking 5-1 file exclusion 5-11 rules file 5-8 violations 5-16

#### Ρ

Polyspace Client for C/C++ installation 1-11 license 1-11 Polyspace In One Click

active project 3-23 overview 3-23 sending files to Polyspace software 3-25 starting verification 3-25 use 3-23 Polyspace products for C components 1-5 installation 1-11 licenses 1-11 overview 1-2 related products 1-16 user interface 1-5 workflow 1-12 Polyspace Queue Manager Interface. See Spooler Polyspace Server for C/C++ installation 1-11 license 1-11 Polyspace verification environment opening 2-5 ppm. See Polyspace project model file preferences assistant configuration 4-24 **Project Manager** default server mode 3-10 server detection 3-21 procedural entities view 4-4 product overview 1-2 progress bar Project Manager window 3-13 3-32 project creation 2-3 2-7 definition 2-3 file types configuration file 2-3 desktop file 2-3 Polyspace project model file 2-3 folders includes 2-4 results 2-4 sources 2-4

opening 3-4 saving 2-14 Project Manager monitoring verification progress 3-13 3-32 opening 2-5 overview 2-5 perspective 2-5 starting verification on client 3-30 starting verification on server 3-10 viewing logs 3-13 3-32 window progress bar 3-13 3-32 Project Manager perspective 1-5 project model file. See Polyspace project model file

## R

related products 1-16 Polyspace products for linking to Models 1-16 Polyspace products for verifying Ada code 1-16 Polyspace products for verifying C++ code 1-16 reports generation 4-31 results downloading from server 3-19 opening 4-3 report generation 4-31 reviewing 4-1 rte view. See procedural entities view Run-time checks perspective call tree view 4-4 coding review progress view 4-4 procedural entities view 4-4 selected check view 4-4 source code view 4-4 variables view 4-4 **Run-Time Checks perspective 1-5** 

opening 4-3 overview 4-4

#### S

selected check view 4-4 server 1-5 3-2 detection 3-21 information in preferences 3-21 installation 1-11 3-21 verification on 3-10 source code view 4-4 Spooler 1-5 monitoring verification progress 3-14 removing verification from queue 3-19 use 3-14 viewing log 3-14

## T

target environment 2-12 troubleshooting failed verification 3-20

#### U

unreachable code example 4-15

#### V

variables view 4-4 verification Ada code 1-16 C code 1-2 C++ code 1-16 client 3-2 compile phase 3-10 failed 3-20 monitoring progress Project Manager 3-13 3-32 Spooler 3-14 phases 3-10 results color-coding 1-2 to 1-3 opening 4-3 report generation 4-31 reviewing 4-1 running 3-2 running on client 3-30 running on server 3-10 starting from Polyspace In One Click 3-2 3-25 from Project Manager 3-2 3-10 3-30 stopping 3-34 troubleshooting 3-20 with MISRA C checking 5-14 Verification stopping 3-33

#### W

workflow basic 1-12 in this guide 1-13