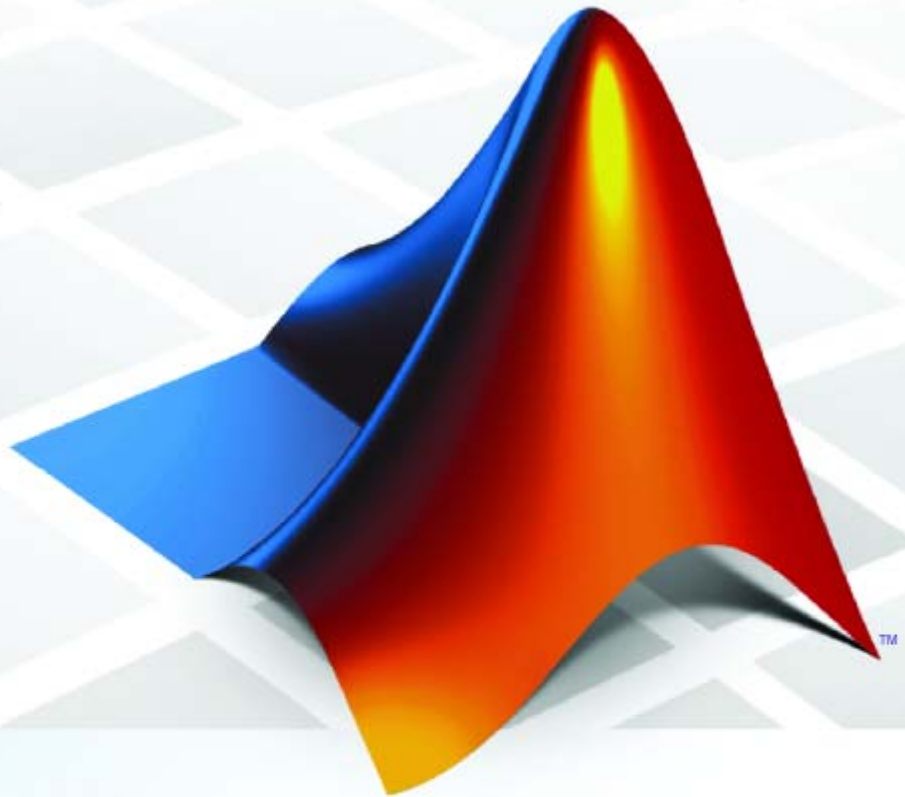


PolySpace[®] Products for C 7

Getting Started Guide



How to Contact The MathWorks



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



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

PolySpace® Products for C Getting Started Guide

© COPYRIGHT 1997–2010 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

The 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)

Introduction to PolySpace Products for Verifying C Code

1

Product Overview	1-2
Ensures Software Reliability	1-2
Decreases Development Time	1-2
Improves the Development Process	1-3
Product Components	1-5
Installing PolySpace Products	1-6
Finding the Installation Instructions	1-6
Obtaining Licenses for PolySpace® Client for C/C++ and PolySpace® Server for C/C++	1-6
Working with PolySpace Software	1-7
Basic Workflow	1-7
The Workflow in This Guide	1-8
Working with PolySpace Project Model Files	1-9
Learning More	1-10
Product Help	1-10
The MathWorks Online	1-10
Related Products	1-11
PolySpace Products for Verifying C++ Code	1-11
PolySpace Products for Verifying Ada Code	1-11
PolySpace Products for Linking to Models	1-11

Setting Up a Project File

2

About This Tutorial	2-2
Overview	2-2
Example Files	2-2
Creating a New Project	2-3
What Is a Project?	2-3
Preparing the Project Folders	2-4
Opening the PolySpace Launcher	2-5
Changing the Default Folder	2-7
Creating a New Project to Verify the Example C File	2-9

Running a Verification

3

About This Tutorial	3-2
Overview	3-2
Before You Start	3-3
Opening the Project	3-4
Using the Launcher to Start a Verification That Runs on a Server	3-5
Starting the Verification	3-5
Monitoring the Progress of the Verification	3-7
Downloading Results from the Server to the Client	3-10
Troubleshooting a Failed Verification	3-12
Using PolySpace In One Click to Start a Verification That Runs on a Server	3-15
Overview of PolySpace In One Click	3-15
Setting the Active Project	3-15
Sending the Files to PolySpace Software	3-17

Using the Launcher to Start a Verification That Runs on a Client	3-24
Starting the Verification	3-24
Monitoring the Progress of the Verification	3-25
Completing the Verification and Stopping the Launcher ..	3-26
Stopping the Verification Before It Completes	3-27

Reviewing Verification Results

4

About This Tutorial	4-2
Overview	4-2
Before You Start	4-2
Opening the Viewer and the Verification Results	4-3
Opening the Viewer	4-3
Selecting the Viewer Mode	4-3
Opening the Results	4-4
Exploring the Viewer Window	4-5
Overview	4-5
Reviewing the Procedural Entities View	4-7
Reviewing Results in Expert Mode	4-10
What Is Expert Mode?	4-10
Switching to Expert Mode	4-10
Reviewing Checks in Expert Mode	4-10
Reviewing Additional Examples of Checks	4-16
Filtering the Types of Checks That You See	4-21
Reviewing Results in Assistant Mode	4-27
What Is Assistant Mode?	4-27
Switching to Assistant Mode	4-27
Selecting the Methodology and Criterion Level	4-28
Exploring Methodology for C	4-28
Reviewing Checks	4-30
Defining a Custom Methodology	4-32

Automatically Testing Unproven Code	4-34
Generating Reports of Verification Results	4-35
PolySpace Report Generator Overview	4-35
Generating Report for example.c	4-36

Checking MISRA C Compliance

5

About This Tutorial	5-2
Overview	5-2
Before You Start	5-2
Setting Up MISRA C Checking	5-3
Opening the Example Project	5-3
Setting the MISRA C Checking Option	5-4
Creating a MISRA C Rules File	5-4
Excluding Files from the MISRA C Checking	5-8
Configuring Text and XML Editors	5-8
Saving the Project with a New Name	5-10
Running a Verification with MISRA C Checking	5-11
Starting the Verification	5-11
Examining the MISRA C Log	5-12
Opening MISRA-C Report	5-15

Using a PolySpace Project Model File

6

About This Tutorial	6-2
Overview	6-2
Before You Start	6-2
Creating a New PolySpace Project Model File	6-3
What Is a PolySpace Project Model File?	6-3

Creating the PolySpace Project Model File	6-3
Creating a Configuration File from a PolySpace Project	
Model File	6-10
Why You Must Have a Configuration File	6-10
Opening the Project Model File	6-10
Entering Additional Required Information	6-11
Saving the Configuration File	6-11
Deleting a Generic Target from the Preferences	
Understanding the Generic Targets Preference	6-13
Deleting the Generic Target Added in This Tutorial	6-13

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 1-6
- “Working with PolySpace Software” on page 1-7
- “Learning More” on page 1-10
- “Related Products” on page 1-11

Product Overview

In this section...
“Ensures Software Reliability” on page 1-2
“Decreases Development Time” on page 1-2
“Improves the Development Process” on page 1-3

Ensures Software Reliability

You can ensure the reliability of your C applications by using PolySpace® verification software to prove code correctness and identify run-time errors. Using advanced verification techniques, PolySpace software performs an exhaustive verification of your source code.

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

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

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

You can also improve the quality of your code by using PolySpace verification software to check that your code complies with MISRA C® standards.¹

Decreases Development Time

Using PolySpace verification 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,

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

but 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.

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. You will spend less time debugging because you can see the exact location of an error in the source code. After you fix errors, you can easily run the verification again.

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

Reviewing the code that might have errors (orange code) can be time-consuming, but PolySpace software helps you with the review process. You can use filters to focus on certain types of errors or you can allow the software to identify the code that you should review.

Improves 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 can check overall reliability of an application.
- Managers can monitor application reliability by generating reports from the verification results.

Product Components

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

PolySpace® Client™ for C/C++
PolySpace® Server™ for C/C++

The user interface includes:

- The *Launcher* for setting up verification parameters and starting verifications.
- The *Viewer* for reviewing verification results.
- *Spooler* for managing verifications that run on a server and downloading results from a server to a client.

Installing PolySpace Products

In this section...
“Finding the Installation Instructions” on page 1-6
“Obtaining Licenses for PolySpace® Client for C/C++ and PolySpace® Server for C/C++” on page 1-6

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++

See “PolySpace License Installation” in the *PolySpace Installation Guide* for information about obtaining licenses for PolySpace products.

Working with PolySpace Software

In this section...

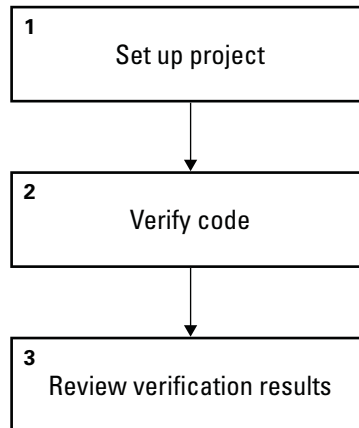
“Basic Workflow” on page 1-7

“The Workflow in This Guide” on page 1-8

“Working with PolySpace Project Model Files” on page 1-9

Basic Workflow

The basic workflow for using PolySpace software to verify C source code is:



In this workflow, you:

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

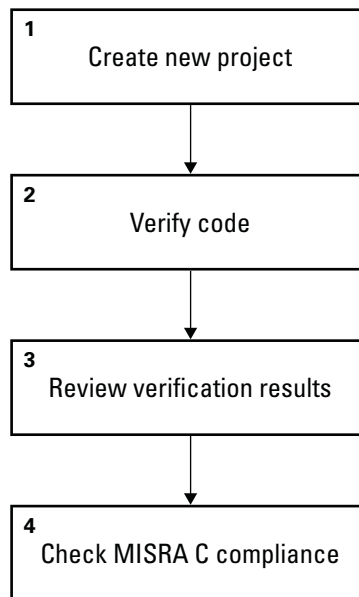
You can use the Launcher to start the verification or you can select files from a Microsoft® Windows® folder and send them to PolySpace software for verification. For verifications that run on a server, you use the Spooler to

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.²

3 Use the Viewer to review verification results.

The Workflow in This Guide

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



In this workflow, you:

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

This step is in the tutorial Chapter 2, “Setting Up a Project File”.

2 Verify a single C file.

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

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

- Using the Launcher to start a verification that runs on a server.
- Using PolySpace In One Click to send files to a server for verification.
- Using the Launcher to start a verification that runs on a client.

3 Review the verification results.

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

Working with PolySpace Project Model Files

A PolySpace project model file is a project file that includes generic target processor information. You can use this file to share project information, but you cannot use it to run a verification. The tutorial Chapter 6, “Using a PolySpace Project Model File” shows you how to work with PolySpace project model files.

Learning More

In this section...
“Product Help” on page 1-10
“The MathWorks Online” on page 1-10

Product Help

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

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

[/www.mathworks.com/access/helpdesk/help/toolbox/polyspace/polyspace.html](http://www.mathworks.com/access/helpdesk/help/toolbox/polyspace/polyspace.html)

The MathWorks Online

For additional information and support, see:

www.mathworks.com/products/polyspace

Related Products

In this section...
“PolySpace Products for Verifying C++ Code” on page 1-11
“PolySpace Products for Verifying Ada Code” on page 1-11
“PolySpace Products for Linking to Models” on page 1-11

PolySpace Products for Verifying C++ Code

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

<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, see the following:

<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, see the following:

<http://www.mathworks.com/products/polyspacemodels1/>

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

Setting Up a Project File

- “About This Tutorial” on page 2-2
- “Creating a New Project” on page 2-3

About This 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 the Project Folders” on page 2-4.

Creating a New Project

In this section...
“What Is a Project?” on page 2-3
“Preparing the Project Folders” on page 2-4
“Opening the PolySpace Launcher” on page 2-5
“Changing the Default Folder” on page 2-7
“Creating a New Project to Verify the Example C File” on page 2-9

What Is a Project?

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

- The location of source files and include folders
- The location of a folder for verification results
- Analysis options

You can create your own project or use an existing project. You create and modify a project using the Launcher graphical user interface.

A project file has one of the following file types:

Project Type	File Extension	Description
Configuration	cfg	Required for running a verification. Does not include generic target processors.
PolySpace Project Model	ppm	For populating a project with analysis options, including generic target processors.
Desktop	dsk	In earlier versions of PolySpace software, for running a verification on a client computer.

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

Preparing the 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`
 - `results`
- 3 Copy the file `example.c` from

`Install_folder\Examples\Demo_C_Single-File\sources`

to

`polyspace_project\sources`

where *Install_folder* is the installation folder.

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

`Install_folder\Examples\Demo_C_Single-File\sources`

to

`polyspace_project\includes`.

Opening the PolySpace Launcher

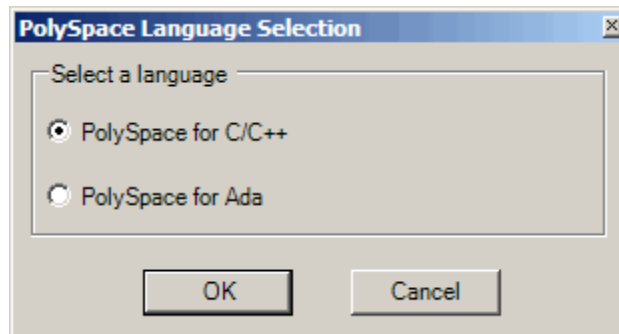
You use the PolySpace Launcher, a graphical user interface, to create a project and start a verification.

To open the PolySpace Launcher:

- Double-click the PolySpace Launcher icon.

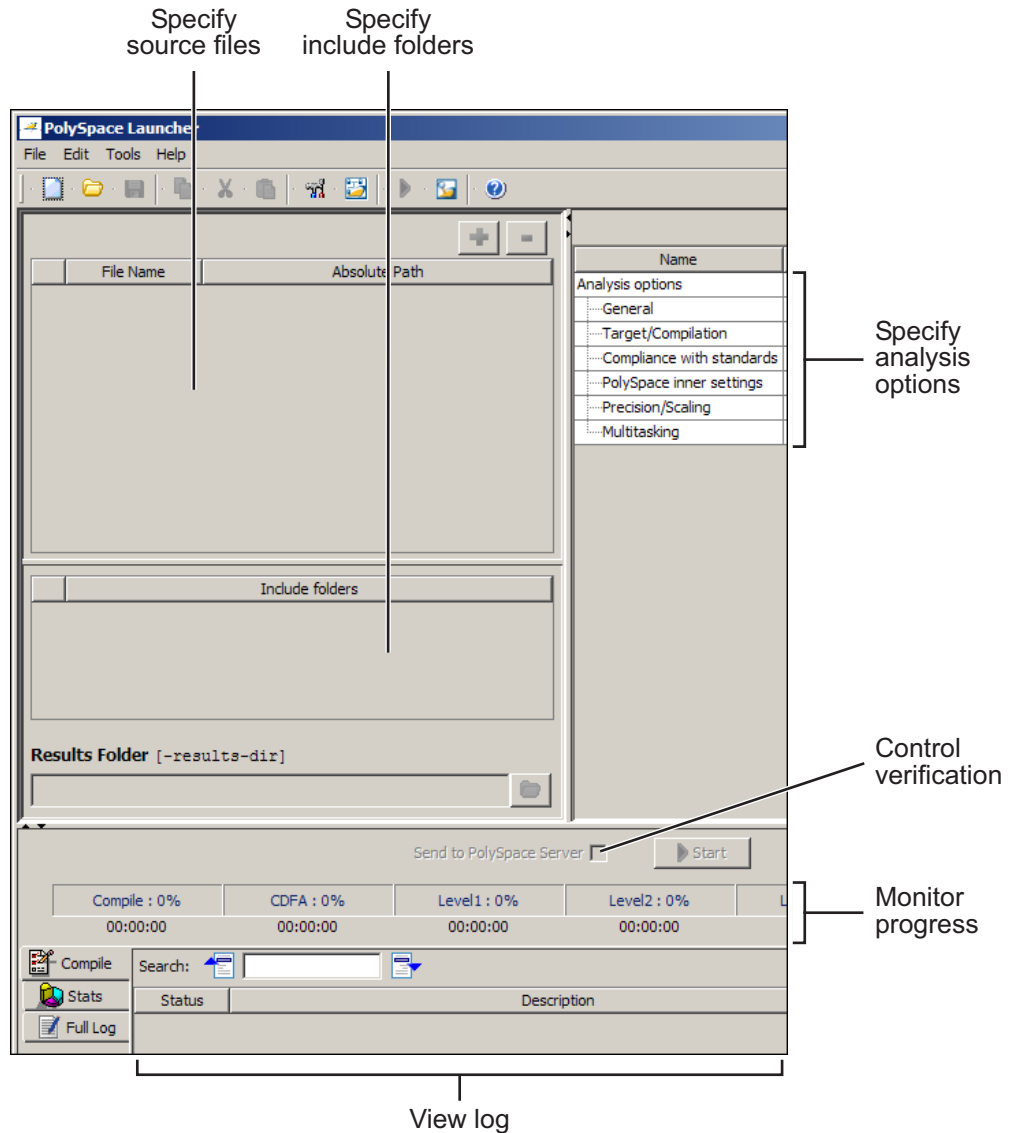


- If you have only PolySpace Client for C/C++ software installed on your computer, skip this step. If you have both PolySpace Client for C/C++ and PolySpace Client for Ada products on your system, the **PolySpace Language Selection** dialog box will appear.



Select **PolySpace for C/C++** and click **OK**.

The PolySpace Launcher window appears:



The Launcher window has three main sections.

Use this section...	For...
Upper-left	Specifying: <ul style="list-style-type: none">• Source files• Include folders• Results folder
Upper-right	Specifying analysis options
Lower	Controlling and monitoring a verification

You can resize or hide any of these sections. You learn more about the Launcher window later in this tutorial.

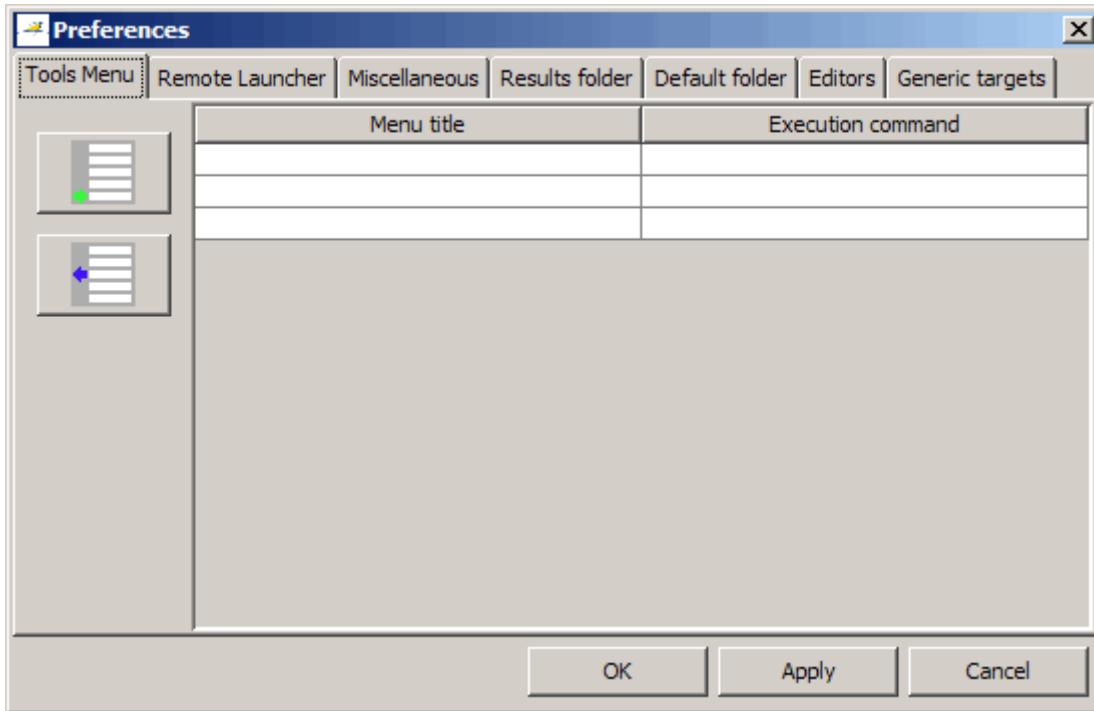
Changing the Default Folder

PolySpace software allows you to specify the default folder that appears in directory browsers in dialog boxes. If you do not change the default folder, the default folder is the installation folder. In this tutorial, you change the default folder to the project folder that you created in “Preparing the Project Folders” on page 2-4. Changing the default folder to the project folder makes it easier for you to locate and specify source files and include folders in dialog boxes.

To change the default folder to the project folder:

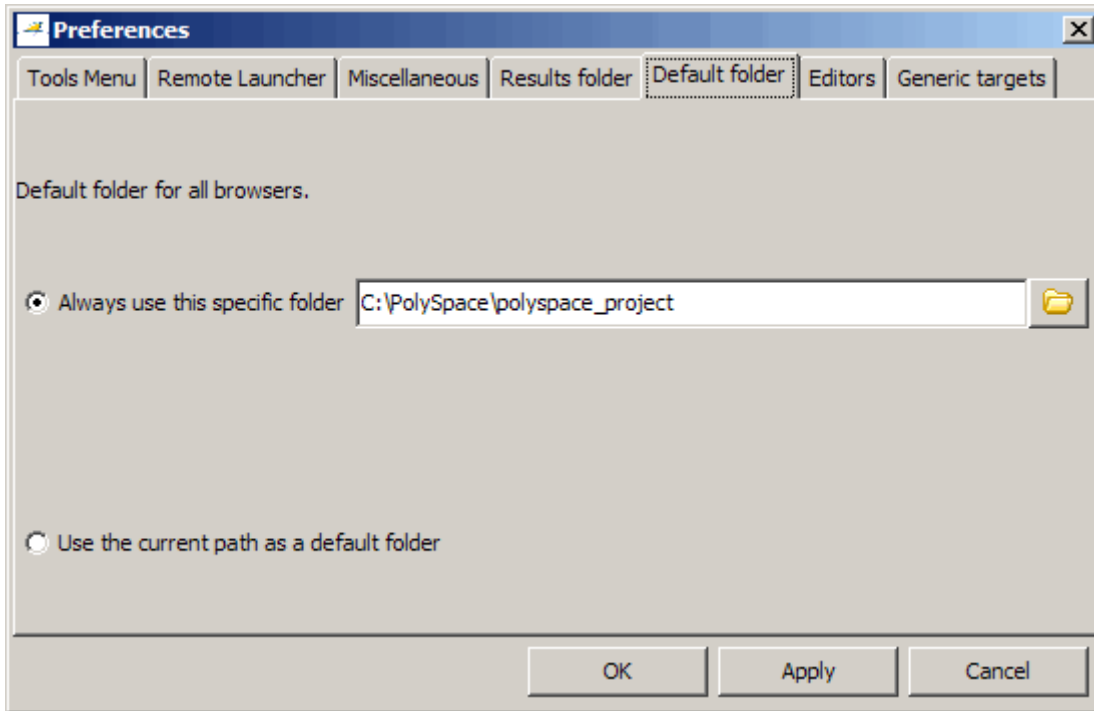
- 1 Select **Edit > Preferences**.

The **Preferences** dialog box appears.



- 2** Select the **Default folder** tab.
- 3** Select **Always use this specific folder** if it is not already selected.
- 4** Enter or navigate to the project folder that you created earlier. In this example, the project folder is `C:\PolySpace\polyspace_project`.

The **Preferences** dialog box should now look like the following.



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

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:

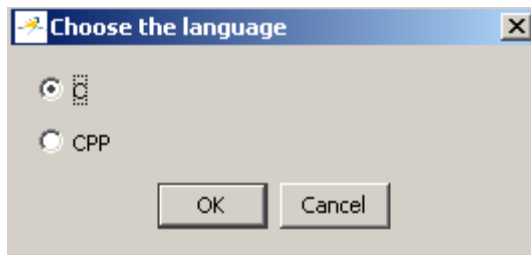
- “Opening a New project” on page 2-10
- “Specifying the Source Files, Include Folders, and Results Folder” on page 2-11
- “Specifying the Analysis Options” on page 2-14
- “Saving the Project” on page 2-16

Opening a New project

To open a new project for verifying `example.c`:

- 1 Select **File > New Project**.

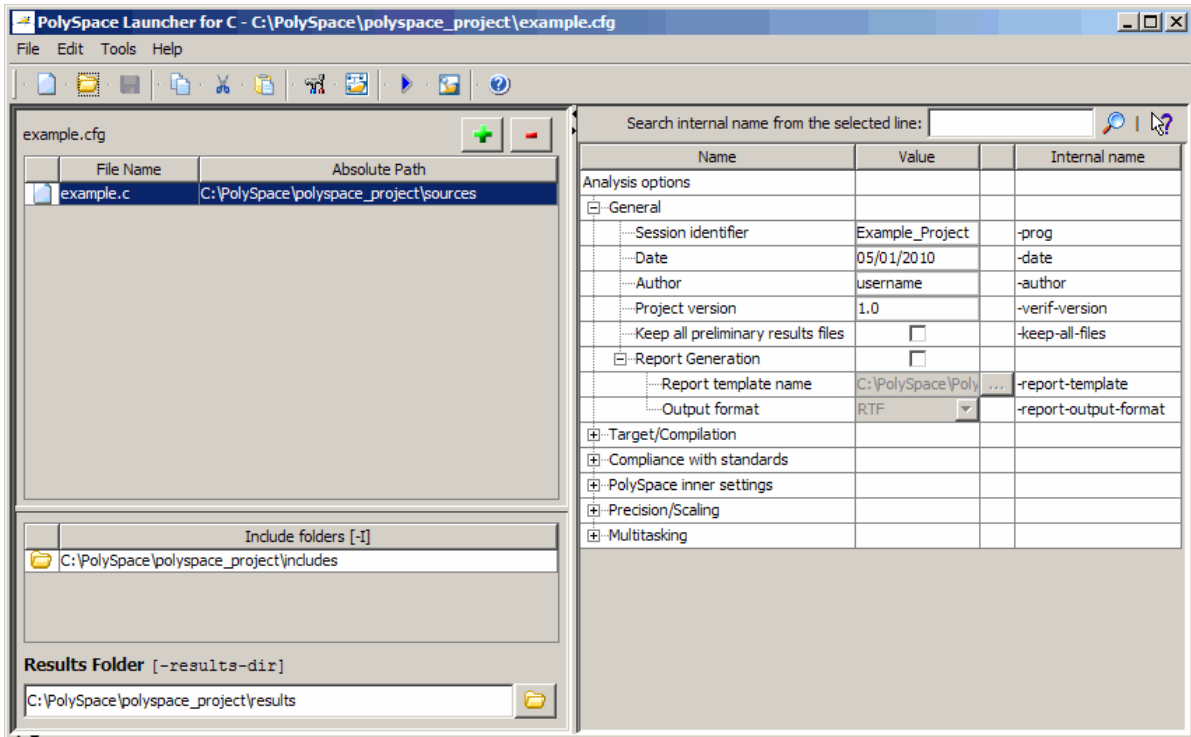
The **Choose the language** dialog box appears:



- 2 Select **C**, then click **OK**.

The default project name, `New_Project`, appears in the title bar.

In the **Analysis options** section, the **General** options node expands with default project identification information and options.



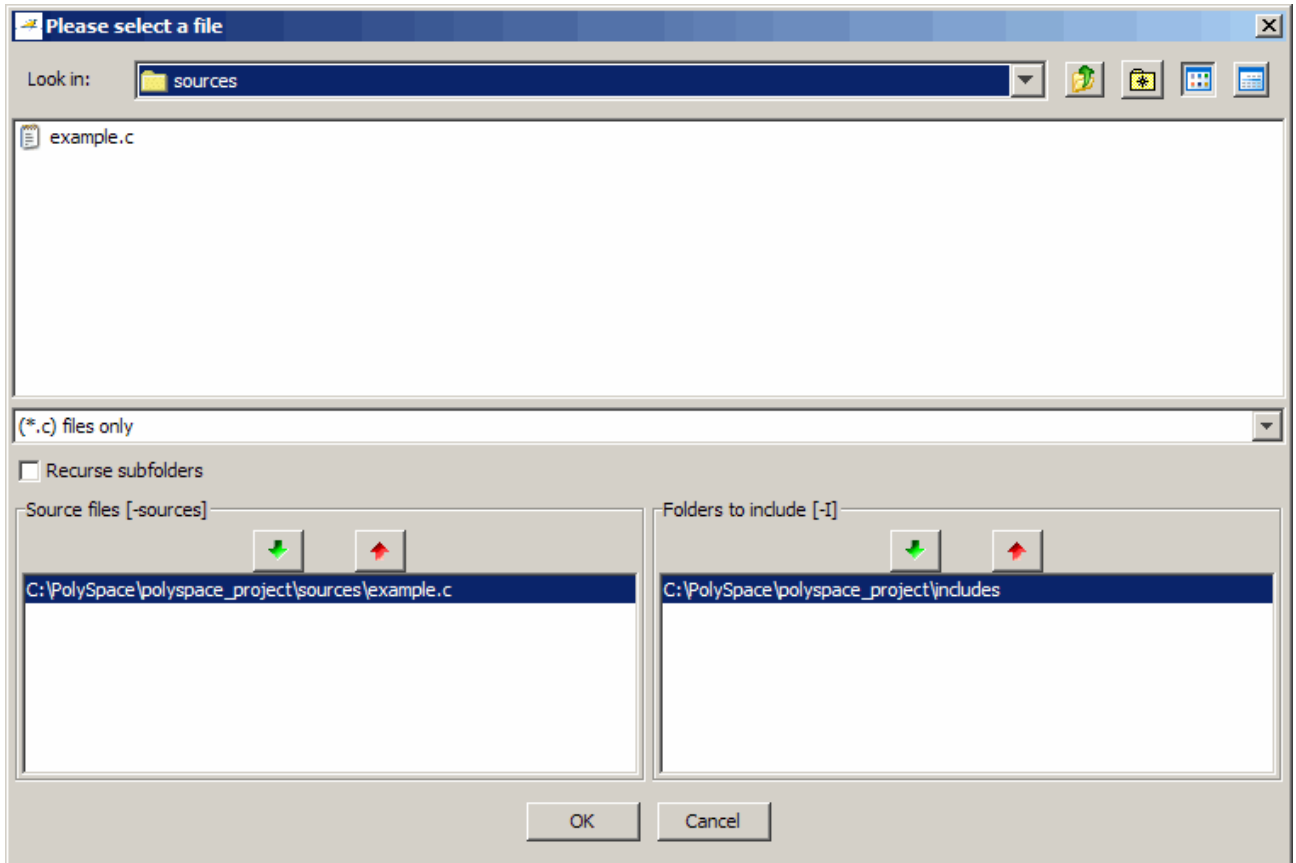
Specifying the Source Files, Include Folders, and Results Folder

To specify the source files, include folders, and results folder for the verification of `example.c`:

- 1 Click the green plus sign button in the upper right of the files section of the Launcher window.



The **Please select a file** dialog box appears.



- 2 The project folder `polyspace_project` should appear in **Look in**. If it does not, navigate to that folder.
- 3 Select the folder `includes` and then click the green down arrow button in the **Folders to include** section.



The path for the folder appears in the source files list.

- 4 Double-click the folder `sources`.

- 5 Select the file `example.c` and then click the green down arrow button in the **Source files** section.

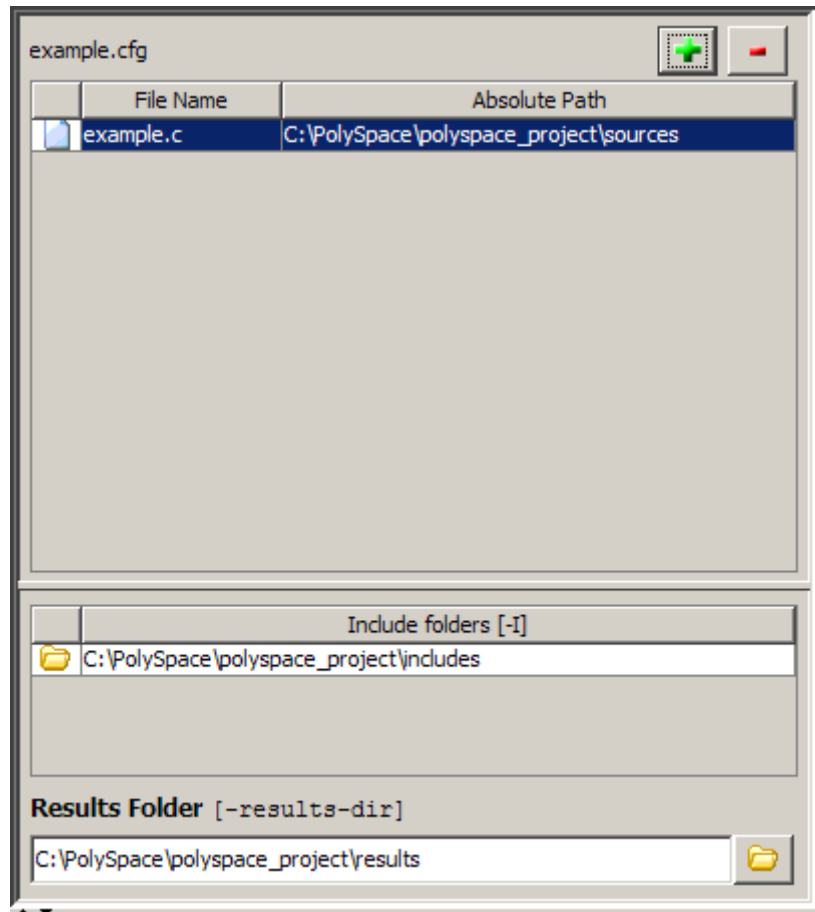


The path for `example.c` appears in the source files list.

Tip You can also drag folder and file names from an open folder directly to the source files list or include list.

- 6 Click **OK** to apply the changes and close the dialog box.
- 7 In **Results Folder**, specify the folder for the verification results. Enter the path for the results folder that you created earlier. In this example, the results folder is `C:\PolySpace\polyspace_project\results`.

The files section in the upper left of the Launcher window now looks like:



Specifying the Analysis Options

The analysis options in the upper-right section of the Launcher window include identification information and parameters that PolySpace software uses during the verification process. For more information about analysis options, see “Options Description” in the *PolySpace Products for C Reference*.

To specify the analysis options for this tutorial:

- 1** In the **General** section, change the **Session identifier** to `Example_Project`.

Note The session identifier cannot contain spaces.

- 2** Expand the **Target/Compilation** section.
- 3** Select **no-predefined-OS** from the drop-down menu next to **Operating system target for PolySpace stubs**.
- 4** Keep the default values for all other options.

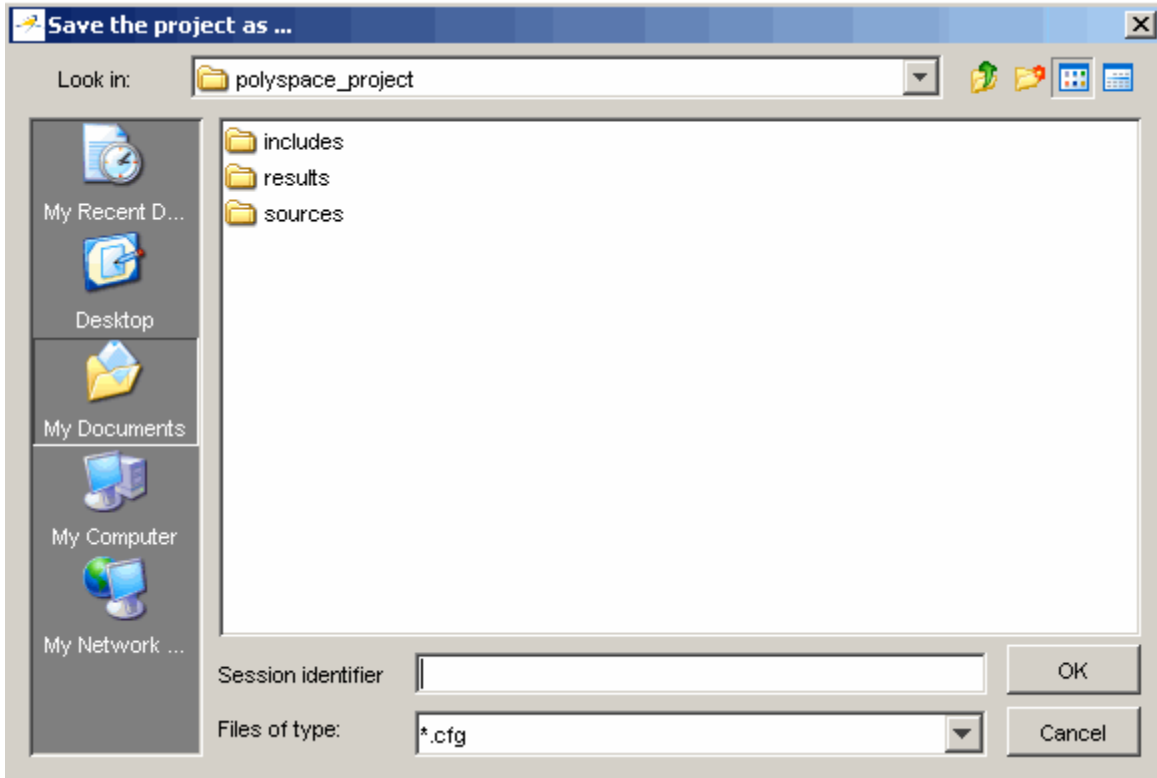
The analysis options will now look like this.

Search internal name from the selected line:		
Name	Value	Internal name
Analysis options		
[-] General		
... Session identifier	Example_Project	-prog
... Date	05/01/2010	-date
... Author	username	-author
... Project version	1.0	-verif-version
... Keep all preliminary results files	<input type="checkbox"/>	-keep-all-files
[-] Report Generation		
... Report template name	C:\PolySpace\PolySpa...	-report-template
... Output format	RTF	-report-output-format
[-] Target/Compilation		
... Target processor type	sparc	-target
... Operating system target for PolySpace stubs	no-predefined-OS	-OS-target
... Defined Preprocessor Macros		-D
... Undefined Preprocessor Macros		-U
... Include		-include
... Command/script to apply to preprocessed files		-post-preprocessing-command
... Command/script to apply after the end of the		-post-analysis-command
[+] Compliance with standards		
[+] PolySpace inner settings		
[+] Precision/Scaling		
[+] Multitasking		

Saving the Project

To save the project:

- 1 Select **File > Save project**. The **Save the project as** dialog box appears.



- 2** In **Look in**, leave the default folder, `polyspace_project`.
- 3** In **Session identifier**, enter `example`.
- 4** In **Files of type**, leave the default `*.cfg`. You must have a project file with type `cfg` to run a verification.

Note You can also run a verification with a project file of type `dsk`. Older versions of PolySpace software created files with type `dsk` for use with a verification running on a desktop PC. For more information about the `dsk` file type, see “What Is a Project?” on page 2-3.

- 5** Click **OK** to save the project and close the dialog box.

Running a Verification

- “About This Tutorial” on page 3-2
- “Opening the Project” on page 3-4
- “Using the Launcher to Start a Verification That Runs on a Server” on page 3-5
- “Using PolySpace In One Click to Start a Verification That Runs on a Server” on page 3-15
- “Using the Launcher to Start a Verification That Runs on a Client” on page 3-24

About This 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	<ul style="list-style-type: none">• Best performance• Large files (more than 800 lines of code including comments)• Multitasking
Client	<ul style="list-style-type: none">• An alternative to the server when the server is busy• Small files with no multitasking <hr/> <p>Note Verification on a client takes more time. You might not be able to use your client computer when a verification is running on it.</p> <hr/>

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

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

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

- 1 The Launcher to start a verification that runs on a server.
- 2 PolySpace In One Click to start a verification that runs on a server.
- 3 The Launcher to start a verification that runs on a client.

Each verification stores the same results in `polyspace_project\results`. 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 Project File”. You use the folders and project file, `example.cfg`, from that tutorial.

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 Project File”. Open `example.cfg` if it is not already open.

To open `example.cfg`:

- 1 If the PolySpace Launcher is not already open, open it by double-clicking the PolySpace Launcher icon.

- 2 Select **File > Open project**.

The **Please select a file** dialog box opens.

- 3 In **Look in**, navigate to `polyspace_project`.

- 4 Select `example.cfg`.

- 5 Click **Open** to open the file and close the dialog box.

Using the Launcher to Start a Verification That Runs on a Server

In this section...

“Starting the Verification” on page 3-5

“Monitoring the Progress of the Verification” on page 3-7

“Downloading Results from the Server to the Client” on page 3-10

“Troubleshooting a Failed Verification” on page 3-12

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 next to the **Start** button in the middle of the Launcher window.



Note If you select **Set this option to use the server mode by default in every new project** in the Remote Launcher pane of the preferences, the **Send to PolySpace Server** check box is selected by default when you create a new project.

- 2 Click **Start**.

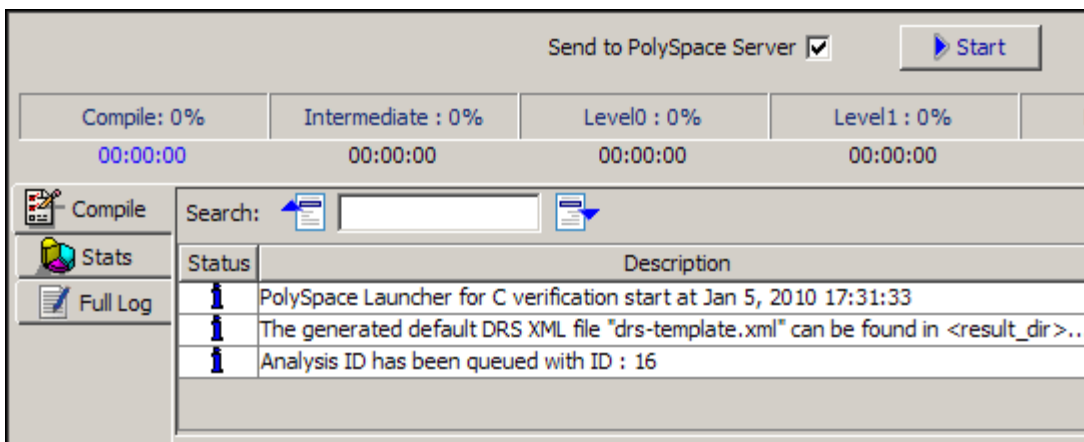
Note If you see the message **Verification process failed**, click **OK** and go to “Troubleshooting a Failed Verification” on page 3-12.

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® standards.
- b Generating a main if it does not find a main and the **Generate a Main** option is selected. For more information about generating a main, see “MAIN GENERATOR OPTIONS (-main-generator) for PolySpace Software” in the *PolySpace Products for C Reference*.
- c Analyzing the code for run-time errors and generating color-coded diagnostics.

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

- A message dialog box tells you that the verification completed. This message means 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 log area tells you 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.



- 3 When you see the message Verification process completed, click **OK** to close the message dialog box.
- 4 Click on any message in the log to get details about the message.

5 Stop the Launcher by clicking **File > Quit**.

Monitoring the Progress of the Verification

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:



The **PolySpace Queue Manager Interface** opens.

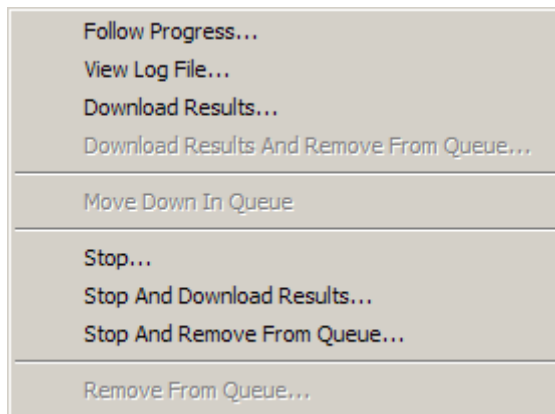
PolySpace Queue Manager Interface							
ID	Author	Application	Results folder	CPU	Status	Date	Language
4	PolySpace	Demo_C	C:\PolySpace\PolySpaceForCandCPP_...	runstr...	completed	14-Dec-2009,...	C
5	polyspace	Demo_C_Single_File	C:\PolySpace\PolySpaceForCandCPP_...	runstr...	completed	14-Dec-2009,...	C
8	PolySpace	Demo_C	C:\PolySpace\polyspace_project\results		completed	17-Dec-2009,...	C
15	username	Example_Project	C:\PolySpace\polyspace_project\results	runstr...	running	05-Jan-2010, ...	C

Connected to Queue Manager localhost User mode

Tip You can also open the Polyspace Queue Manager Interface by clicking the PolySpace Queue Manager icon  in the PolySpace Launcher toolbar.

2 Point anywhere in the row for ID 1.

3 Right-click to open the context menu for this verification.



4 Select **View log file**.

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

```
C:\PolySpace\PolySpace_Common\RemoteLauncher\wbin\psqueue-progress.exe
GUI files generation complete.
Generating remote file
Done

Certain (red) errors have been detected in the analysed code during the verification phase.
Analysis continuing because the option -continue-with-red-errors was used.

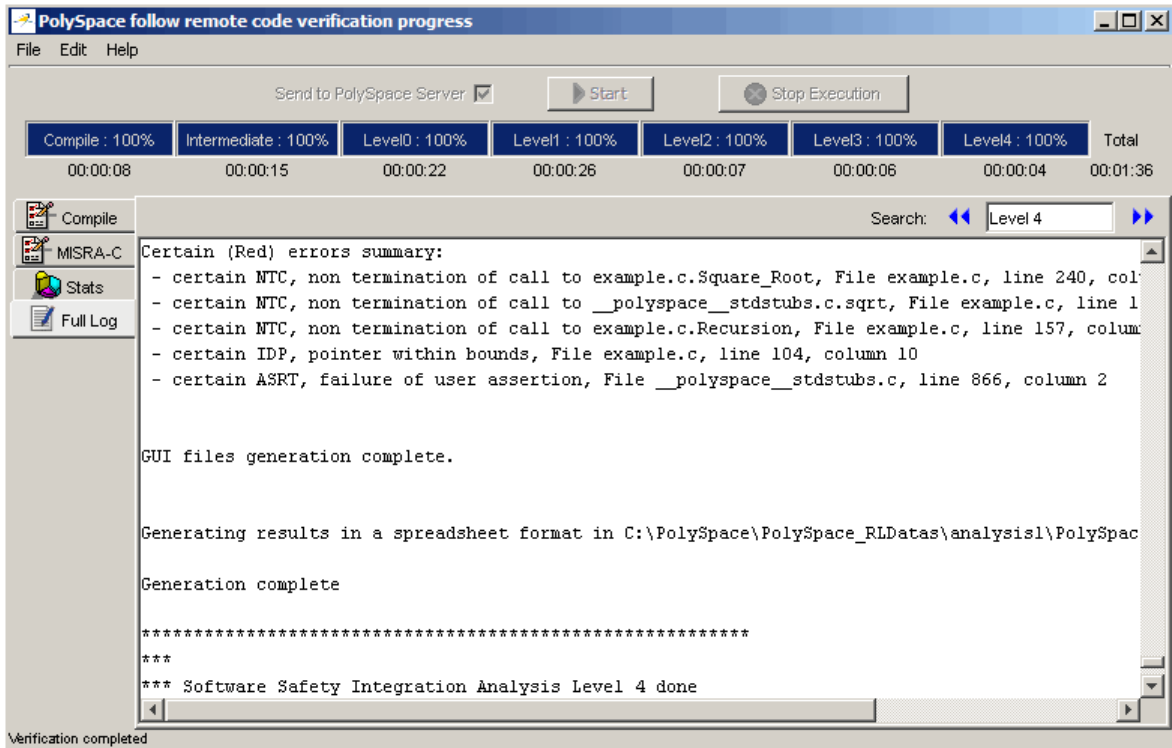
*****
***
*** Level 4 Software Safety Analysis done
***
*****
Ending at: Apr 11, 2008 12:29:8
User time for pass4: 35.8real, 35.8u + 0s
User time for polyspace-c: 176.5real, 176.5u + 0s

***
*** End of PolySpace Verifier analysis
***
Press enter to close the window ...
```

5 Press **Enter** to close the window.

6 Select **Follow Progress** from the context menu.

A Launcher window labeled **PolySpace follow remote analysis progress for C** appears.



You can monitor the progress of the verification by watching the progress bar and viewing the logs at the bottom of the window. The word **processing** appears under the current phase. The progress bar highlights each completed phase and displays the amount of time for that phase.

The logs report additional information about the progress of the verification. The information appears in the log display area at the bottom of the window. The full log displays by default. It displays messages, errors, and statistics for all phases of the verification. You can search the full log by entering a search term in the **Search in the log** box and clicking the left arrows to search backward or the right arrows to search forward.

- 7 Click the **Compile Log** button 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.
- 8 Click the **Stats** button to display statistics, such as analysis options, stubbed functions, and the verification checks performed.
- 9 Click the refresh button



to update the stats log display as the verification progresses.

- 10 Select **File > Quit** to close the progress window.
- 11 Wait for the verification to complete.

When the verification completes, the status in the **PolySpace Queue Manager Interface** changes from running to completed.

PolySpace Queue Manager Interface							
Operations Help							
ID	Author	Application	Results folder	CPU	Status	Date	Language
4	PolySpace	Demo_C	C:\PolySpace\PolySpaceForCandCPP_...	runstr...	completed	14-Dec-2009,...	C
5	polyspace	Demo_C_Single_File	C:\PolySpace\PolySpaceForCandCPP_...	runstr...	completed	14-Dec-2009,...	C
8	PolySpace	Demo_C	C:\PolySpace\polyspace_project\results		completed	17-Dec-2009,...	C
15	username	Example_Project	C:\PolySpace\polyspace_project\results	runstr...	completed	05-Jan-2010, ...	C

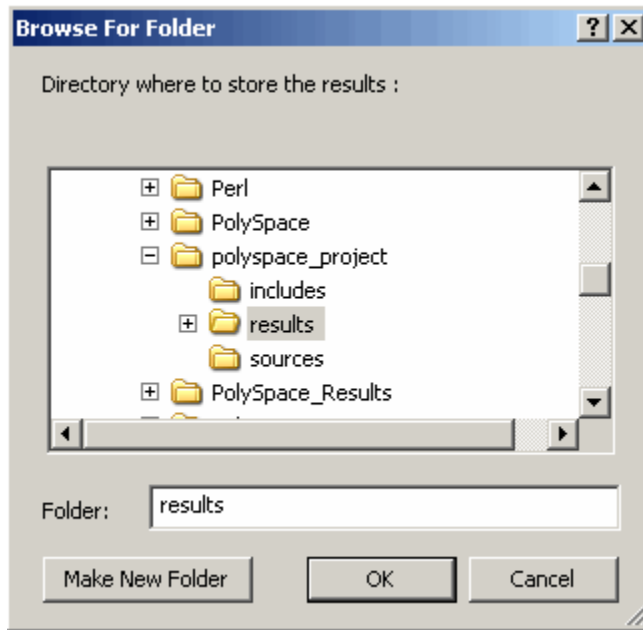
Connected to Queue Manager localhost User mode

Downloading Results from the Server to the Client

At the end of the verification, the results are on the server. To download the results to your client:

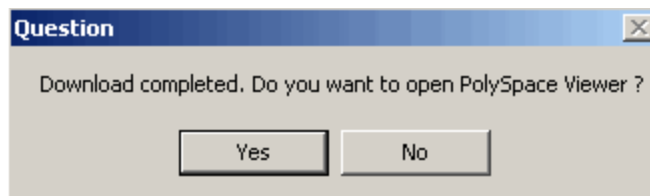
- 1 In the **PolySpace Queue Manager Interface**, select **Download Results** from the context menu for the verification.

The **Browse For Folder** dialog box appears with the `polyspace_project\results` folder selected.



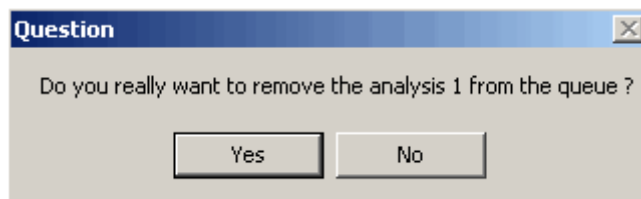
- 2 Click **OK** to close the dialog box.

A dialog box appears telling you that the download is complete and asking if you want to open the PolySpace Viewer.



- 3 Click **No**.
- 4 Select **Remove From Queue** from the context menu.

A dialog box appears asking you to confirm that you want to remove the verification from the queue.



5 Click **Yes**.

Note

- To download the results and remove the verification from the queue, select **Download Results And Remove From Queue** from the context menu.
 - If you download results before the verification completes, you get partial results and the verification continues.
-

6 Select **Operations > Exit** to close the **PolySpace Queue Manager Interface**.

Once the results are on your client, you can review them using the PolySpace Viewer. You review the 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

The verification fails if your computer does not have the minimal hardware requirements. For information about the hardware requirements, see

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.
- Select the **Continue with current configuration option** in the General section of the Analysis options and run the verification again.

You Did Not Specify the Location of Included Files

If you see a message in the log, such as the following, either the files are missing or you did not specify the location of included files.

```
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 2-9.

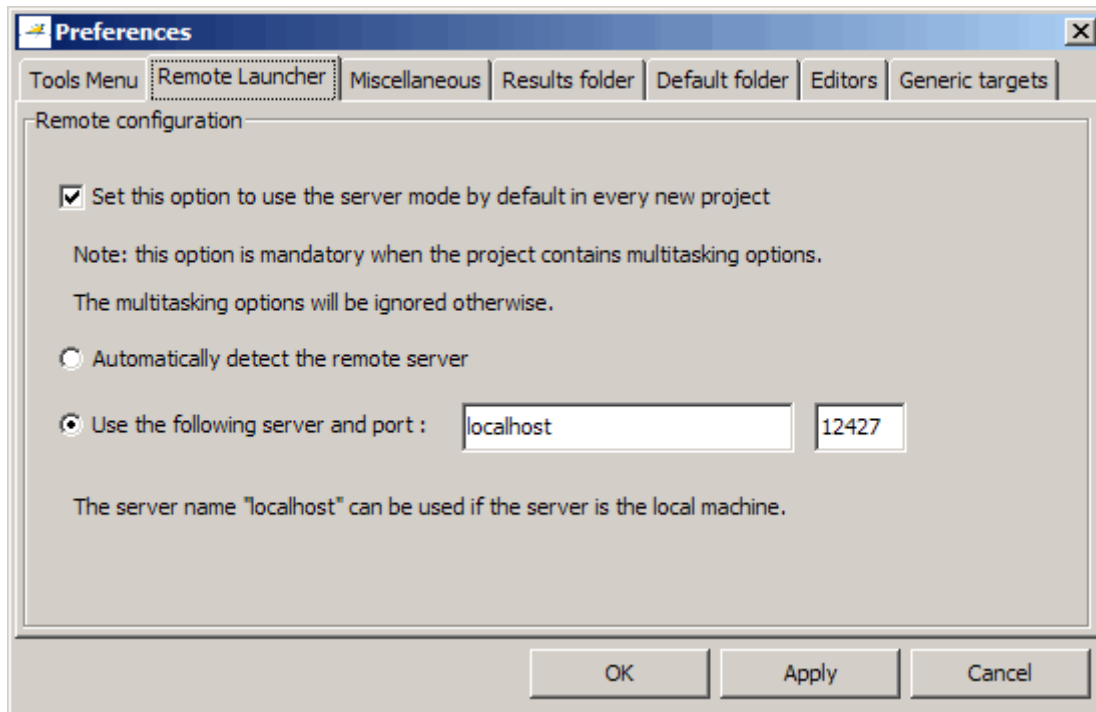
PolySpace Software Cannot Find the Server

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 **Edit > Preferences**.
- 2** Select the **Remote Launcher** tab.



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

Using PolySpace In One Click to Start a Verification That Runs on a Server

In this section...

“Overview of PolySpace In One Click” on page 3-15

“Setting the Active Project” on page 3-15

“Sending the Files to PolySpace Software” on page 3-17

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 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

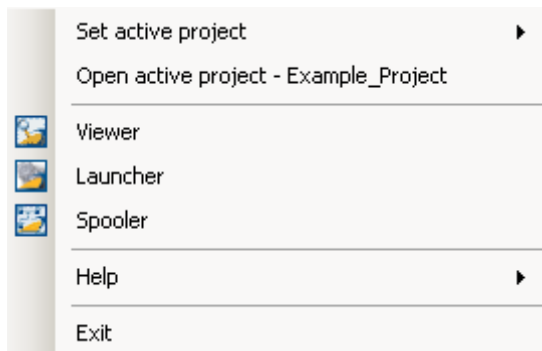
The active project is the project that PolySpace In One Click uses to verify the files that you select. Once you have set an active project, it remains active until you change the active project. PolySpace software uses the analysis options from the project; it does not use the source files or results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

To set the active project:

- 1 Right-click the PolySpace In One Click icon in the taskbar area of your Windows desktop:

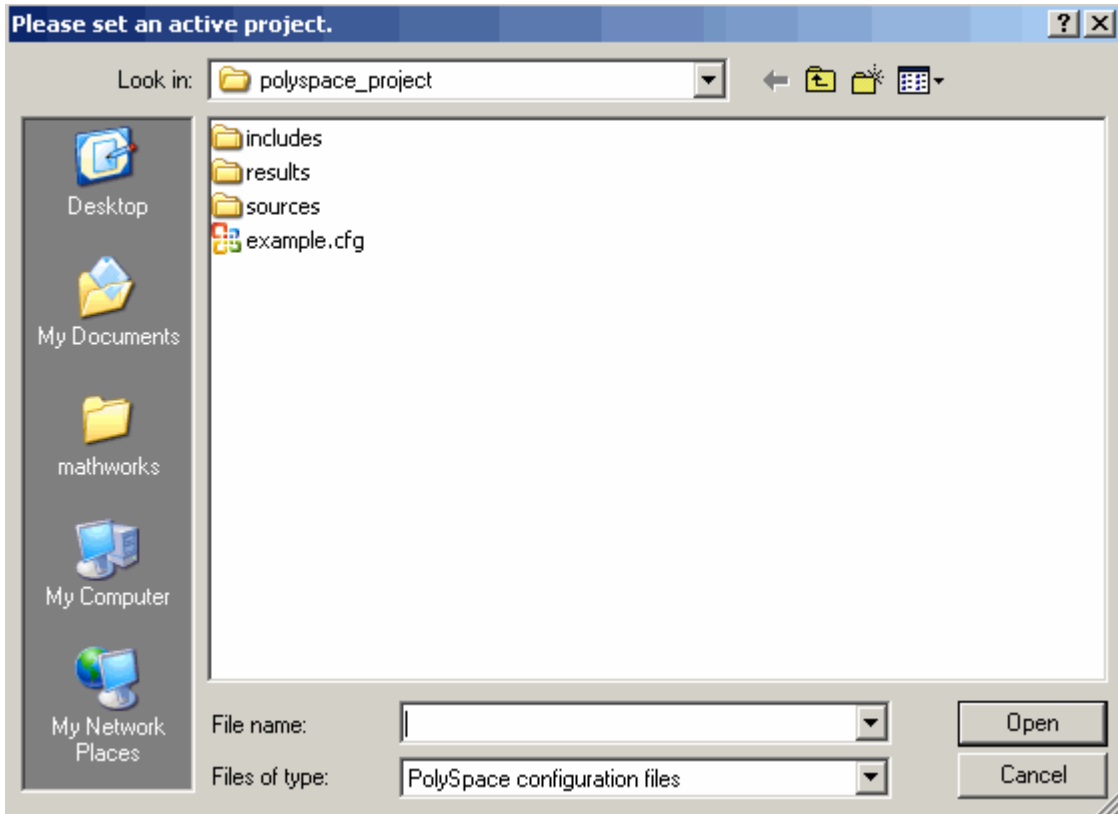


The context menu appears.



2 Select **Set active project > Browse** from the menu.

The **Please set an active project** dialog box appears:



- 3** In **Look in**, navigate to `polyspace_project`.
- 4** Select `example.cfg`.
- 5** Click **Open** to apply the changes and close the dialog box.

Sending the Files to PolySpace Software

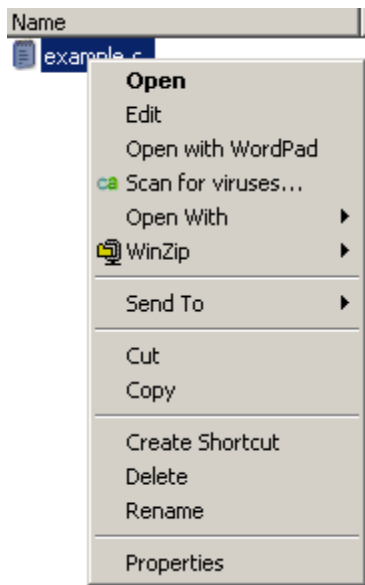
You can send several files to PolySpace software for verification. For this tutorial, you send one file, `example.c`.

To send `example.c` to PolySpace software for verification:

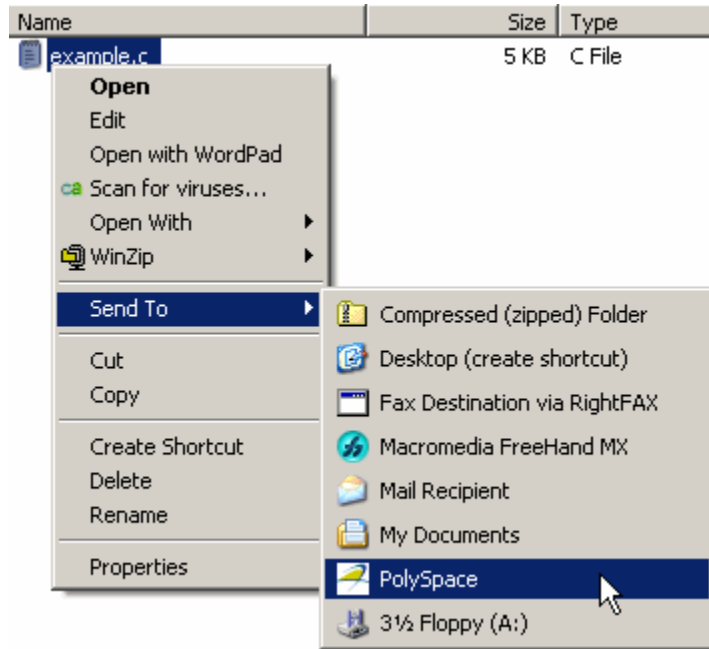
- 1** Navigate to the folder `polyspace_project\sources`.

2 Right-click the file `example.c`.

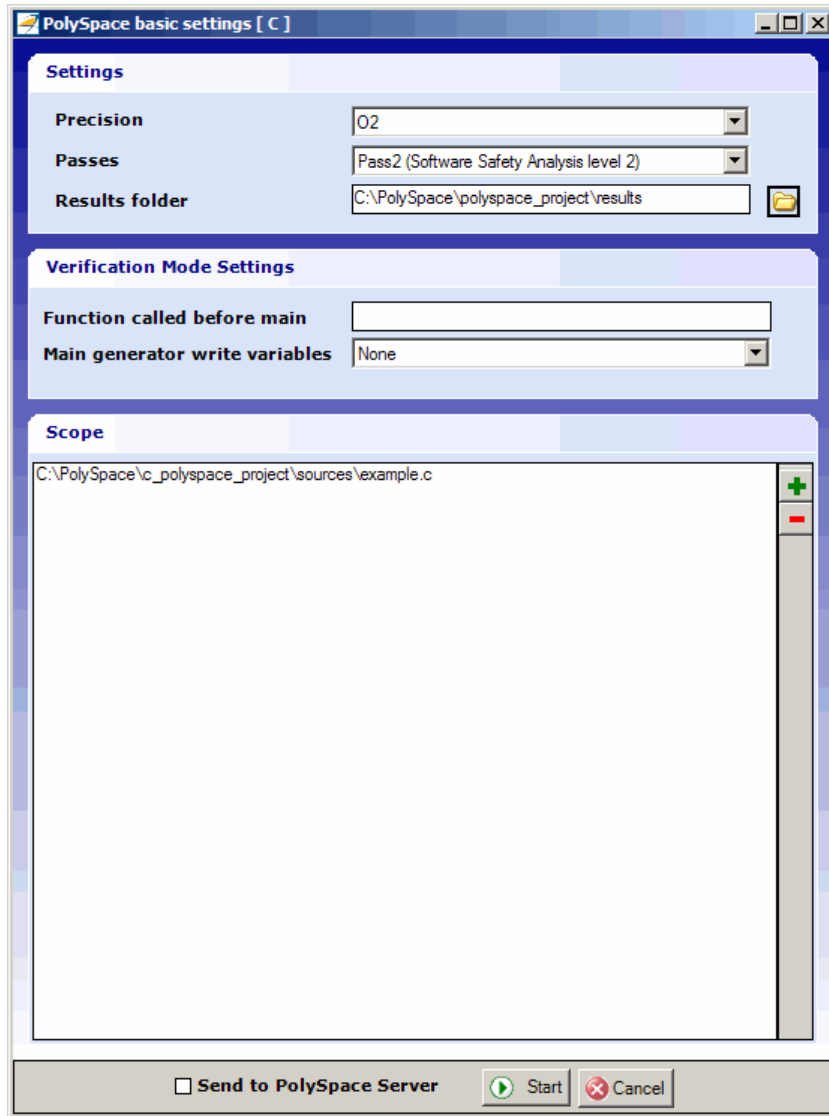
The context menu appears.



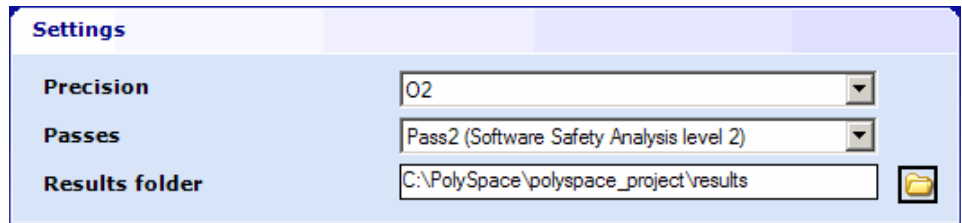
3 Select **Send To > PolySpace**.



The **PolySpace basic settings** dialog box appears.



4 Make sure that **Results folder** is polyspace_project\results.



5 Select the **Send to PolySpace Server** option if it is not already selected.

6 Leave the default values for the other parameters.

Click **Start**.

The verification log appears.

- Monitor the verification using the Spooler. For information on using the Spooler to monitor a verification on a server, see “Monitoring the Progress of the Verification” on page 3-7.
- When the verification completes, download the results to `polyspace_project\results`. For information on downloading results from a server to a client, see “Downloading Results from the Server to the Client” on page 3-10

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

Using the Launcher to Start a Verification That Runs on a Client

In this section...
“Starting the Verification” on page 3-24
“Monitoring the Progress of the Verification” on page 3-25
“Completing the Verification and Stopping the Launcher” on page 3-26
“Stopping the Verification Before It Completes” on page 3-27

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 Open the Launcher if it is not already open.

2 Open the project file `example.cfg` if it is not already open.

For information about opening a project file, see “Opening the Project” on page 3-4.

3 Make sure that the **Send to PolySpace Server** check box is clear.

4 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 only appears when you clear the **Send to PolySpace Server** check box.

5 Click the **Start** button.



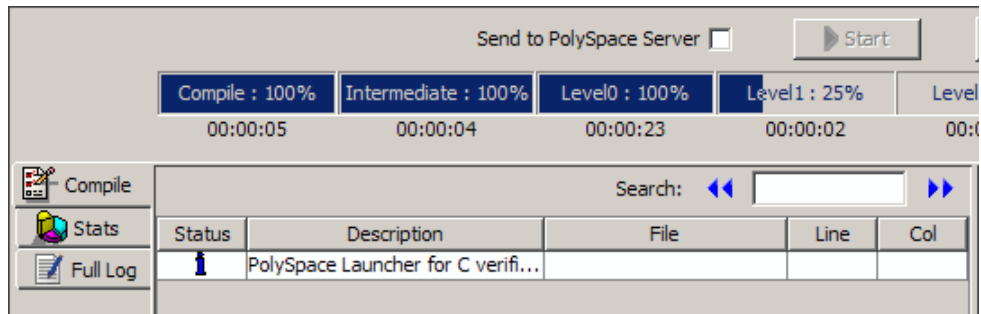
- 6 If you see a caution that PolySpace software will remove existing results from the results folder, click **Yes** to continue and close the message dialog box.

The progress bar and logs area of the Launcher window become active.

Note If you see the message *Verification process failed*, click **OK** and go to “Troubleshooting a Failed Verification” on page 3-12.

Monitoring the Progress of the Verification

You can monitor the progress of the verification by watching the progress bar and viewing the logs at the bottom of the Launcher window.



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

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

- 1 The compile log displays by default.

This log displays 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 **Stats** button to display statistics, such as analysis options, stubbed functions, and the verification checks performed.

3 Click the refresh button



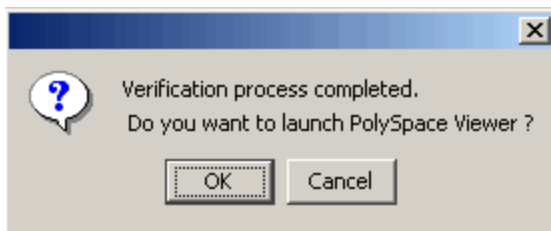
to update the display as the verification progresses.

4 Click the **Full Log** button to display messages, errors, and statistics for all phases of the verification.

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

Completing the Verification and Stopping the Launcher

When the verification completes, a message dialog box appears telling you that the verification is complete and asking if you want to open the Viewer. For this tutorial, do not open the Viewer at this point.



To indicate that you do not want to open the Viewer:

- Click **Cancel**.

You can also open the Viewer from the Launcher toolbar, but for this tutorial, you do not do this. For this tutorial, close the Launcher.

To close the Launcher:

- Select **File > Quit**.

In the tutorial Chapter 4, “Reviewing Verification Results”, you open the Viewer and review the verification results.

Stopping the Verification Before It Completes

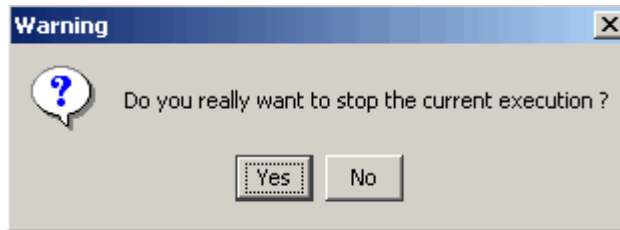
You can stop the verification before it completes. If you stop the verification, results will be incomplete, and if you start another verification, the verification starts over from the beginning.

To stop a verification:

- 1 Click the **Stop Execution** button.



A warning dialog box appears.



- 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 Launcher window does *not* stop the verification. To resume display of the verification progress, open the Launcher window and open the project that you were verifying when you closed the Launcher window.

Reviewing Verification Results

- “About This Tutorial” on page 4-2
- “Opening the Viewer and the Verification Results” on page 4-3
- “Exploring the Viewer Window” on page 4-5
- “Reviewing Results in Expert Mode” on page 4-10
- “Reviewing Results in Assistant Mode” on page 4-27
- “Automatically Testing Unproven Code” on page 4-34
- “Generating Reports of Verification Results” on page 4-35

About This 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.

PolySpace Client for C/C++ provides a graphical user interface, called the Viewer, that you use to review results. In this tutorial, you learn:

- 1 How to use the Viewer, including how to:
 - Open the Viewer and open verification results.
 - Select the Viewer mode.
 - 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\results\RTE_px_02_Example_Project_LAST_RESULTS.rte.
```

Opening the Viewer and the Verification Results

In this section...

“Opening the Viewer” on page 4-3

“Selecting the Viewer Mode” on page 4-3

“Opening the Results” on page 4-4

Opening the Viewer

You use the Viewer to review verification results. Open the Viewer if it is not already open.

To open the Viewer:

- Double-click the PolySpace Viewer icon:



Note You can also open the Viewer from the Launcher by clicking the Viewer icon in the Launcher toolbar with or without an open project.

Selecting the Viewer Mode

You can review verification results in *expert* mode or *assistant* mode:

- In expert mode, you decide how you review the results.
- In assistant mode, PolySpace software guides you through the results.

You switch from one mode to the other by clicking a button in the Viewer toolbar. For this part of the tutorial, the Viewer should be in expert mode. If the Viewer is in expert mode, the mode button in the toolbar displays **Assistant**.



If the Viewer is not in expert mode, click the mode button to switch to expert mode.



You learn more about expert and assistant modes later in this tutorial.

Opening the Results

To open the verification results:

- 1** Select **File > Open**.
- 2** In the **Please select a file dialog box**, navigate to `polyspace_project\results` and select the file `RTE_px_02_Example_Project_LAST_RESULTS.rte`.
- 3** Click the **Open** button.

The results appear in the Viewer window.

Note The file `RTE_px_02_Example_Project_LAST_RESULTS.rte` represents the verification with the highest level of precision. The lower level results files that you see in the `polyspace_project\results` folder represent lower precision verifications.

Exploring the Viewer Window

In this section...
“Overview” on page 4-5
“Reviewing the Procedural Entities View” on page 4-7

Overview

The PolySpace Viewer looks like:

4 Reviewing Verification Results

Coding review progress view

Selected check view

The screenshot displays the PolySpace Viewer interface with several panels:

- Coding review progress:** A table showing the status of the review.
- Procedural entities view:** A tree view of the project's entities.
- Variables view:** A table of variables and their properties.
- Source code view:** The source code for the selected entity.
- Call tree view:** A tree view of the call graph.

Coding review progress		Count	Progress
num NTC reviewed / num NTC to review (Red)	1/3	33	
num reviewed / num to review (Red)	1/5	20	
Software reliability indicator	99/191	51	

Procedural entities	Line	Col	Details	Reviewed	Acronym					
Example_Project	5	77	10	99						
example.c	4	8	83	1						
Close_To_Zero ()	3	10	37	12						
Non_Infinite_Loop ()	11	86	11	100						
Pointer_Arithmetic ()	1	3	1	19	69	12	96	example.c		
RTE ()	1	3	222	5	100			example.c		
Recursion ()	1	14	137	12	93			example.c		
Recursion_caller ()	1	4	151	12	100			example.c		
IRV_0	1	1	152	10				function r...		
NIVL_1	1	1	155	6				local varia...		
NIVL_2	1	1	155	16				local varia...		
NIV_3	1	1	157	5				the exam...		NOW
IRV_4	1	1	182	5				function r...		
Square_Root ()	1	2	4	185	12	100		example.c		
Square_Root_conv ()	8	179	12	100				example.c		
Unreachable_Code ()	3	1	8	199	12	92		example.c		
get_oil_pressure ()	2	2	21	11	50			example.c		
NIVL_0	1	1	25	6				Warning : ...		
ASRT_1	1	1	26	2				Warning : ...		
NIVL_2	1	1	26	10				local varia...		
NIVL_3	1	1	27	9				local varia...		
__polyspace_stdstubs.c	1	69	2	14	1	98		__polyspa...		
__polyspace_main.c	2	1	100					__polyspa...		

Variables	# Read	# Write	W.T.	R.T.	Line	Col	Values
Example_Project							
__polyspace_stdstubs.erno	0	2			175	4	
__polyspace_stdstubs._init					175	4	
__polyspace_stdstubs.sort					896	31	

```
150
151 static void Recursion_caller(void)
152 { int x=random_int();
153
154
155 if ((x>-4) && (x < -1))
156 {
157     Recursion( ex ); // always encounters a division by zero
158 }
159
160
161 x = 10;
162 if (random_int() >= 0)
163 {
```

Calls	Line
Example Recursion_caller	151
pst_stubs_0.random_int	152
example Recursion	157
example Recursion	147
pst_stubs_0.random_int	162
example Recursion	164
example Recursion	147
example RTE	238
__polyspace_m.in.main	51

Procedural entities view

Variables view

Source code view

Call tree view

The appearance of the Viewer toolbar depends on the Viewer mode. Because the Viewer is in expert mode, the expert mode toolbar displays.



In both expert mode and assistant mode, the Viewer window has six sections below the toolbar. Each section provides a different view of the results. The following table describes these views.





This view...	Displays...
Procedural entities view (lower left)	List of the diagnostics (checks) for each file and function in the project
Source code view (lower right)	Source code for a selected check in the procedural entities view
Coding review progress view (upper left)	Statistics about the review progress for checks with the same type and category as the selected check
Selected check view (upper right)	Details about the selected check
Variables view	Information about the global variables declared in the source code Note The file that you use in this tutorial does not have global variables.
Call tree view	Tree structure of function calls

You can resize or hide any of these sections. You learn more about the Viewer window later in this tutorial.






Reviewing the Procedural Entities View


The procedural entities view, in the lower-left part of the Viewer window, displays a table with information about the diagnostics for each file in the project. The procedural entities view is also called the RTE (run-time error)

view. When you first open the results file from the verification of `example.c`, the procedural entities view looks like:

Procedural entities					Line	Col	%	Details	Reviewed	Acronym
Example_Project	5	40	8	13			88		<input type="checkbox"/>	
example.c	4	5	8	13	1		79	example.c	<input type="checkbox"/>	
__polyspace__stdstubs	1	35	2		1		95	__polyspace__stdstubs.c	<input type="checkbox"/>	
__polyspace__main.c					1		0	__polyspace__main.c	<input type="checkbox"/>	

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

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

Tip If you see three dots in place of a heading, , resize the column until you see the heading. Resize the procedural entities view to see additional columns.

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 displays in the other views. In the following examples, you learn how to use the views and how they interact.

Reviewing Results in Expert Mode

In this section...

- “What Is Expert Mode?” on page 4-10
- “Switching to Expert Mode” on page 4-10
- “Reviewing Checks in Expert Mode” on page 4-10
- “Reviewing Additional Examples of Checks” on page 4-16
- “Filtering the Types of Checks That You See” on page 4-21

What Is Expert Mode?

In expert mode, you can see all checks from the verification in the PolySpace Viewer. You decide which checks to review and in what order to review them.

Switching to Expert Mode

If the Viewer is in expert mode, the mode toggle button displays **Assistant**. If the Viewer is in assistant mode, the mode toggle button displays **Expert**. To switch from assistant to expert mode:

- Click the Viewer mode button:



The Viewer window toolbar displays buttons and menus specific to expert mode.

Reviewing Checks in Expert Mode

In this part of the tutorial, you learn how to use the Viewer window views to examine checks from a verification. This part of the tutorial covers:

- “Selecting a Check to Review” on page 4-11
- “Displaying the Calling Sequence” on page 4-12
- “Tracking Review Progress” on page 4-13
- “Tracking Reviewed Checks in Procedural Entities View” on page 4-15

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 window, expand `example.c`.
- 2 Expand the red procedure `Pointer_Arithmetic()`.

A color-coded list of the checks performed on `Pointer_Arithmetic()` appears:

Icon	Acronym	1	2	3	6	89	12	90	example.c
✓	OVFL.2				1	94	23		operation [+] on scalar does not ...
✓	IDP.3				1	98	8		pointer is within its bounds
✓	IRV.6				1	100	5		function returns an initialized value
✓	IRV.7				1	102	9		function returns an initialized value
!	IDP.8	1				104	10		Error : pointer is outside its bounds
✗	UNR.10		1			107	8		unreachable code
✗	OVFL.12		1			108	11		Unreachable check : overflow op...
✓	IRV.13				1	112	6		function returns an initialized value
?	IDP.15			1		114	18		Warning : pointer may be outside...
✓	IDP.22				1	119	8		pointer is within its bounds

Each item in the list of checks 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 about different types of checks, see “Check Descriptions” in the *PolySpace Products for C Reference*.

- 3 Click the red `IDP.8`.

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

```
example.c
92     int i, *p = array;
93
94     for(i = 0; i < 100; i++)
95     {
96         *p = 0;
97         p++;
98     }
99
100    if(get_bus_status() > 0)
101    {
102        if(get_oil_pressure() > 0)
103        {
104            *p = 5; /* Out of bounds */
105        }
106        else
107        {
108            i++;
109        }
110    }
```

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

An error message box displays 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()`:

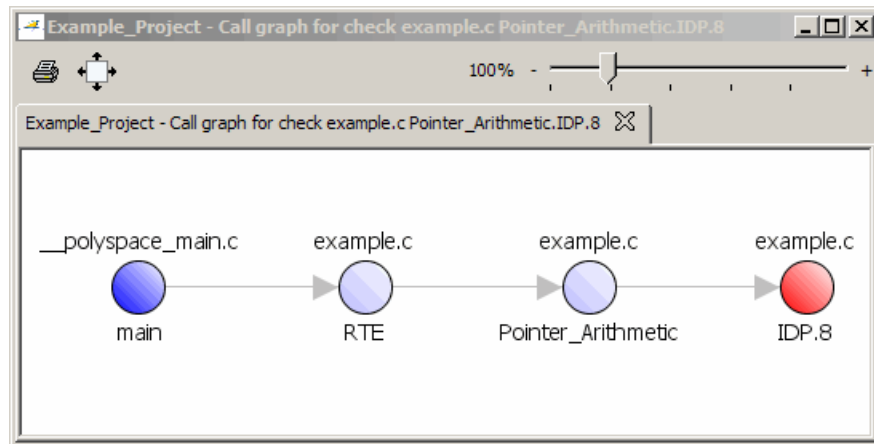
1 Expand `Pointer_Arithmetic()`.

2 Click the red IDP.8.

- 3 Click the call graph button in the 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()`:

- 1 Expand `Pointer_Arithmetic()`.
- 2 Click the red IDP.8.

A table with statistics about the review progress for that category and severity of error appear in the upper-left part of the window.

Coding review progress	Count	Progress
num IDP reviewed / num IDP to review (Red)	0/1	0
num reviewed / num to review (Red)	0/5	0
Software reliability indicator	113/230	49

The **Count** column displays a ratio and the **Progress** column displays the equivalent percentage. The first row displays the ratio of reviewed checks to total checks having the color and category of the current check. In this example, it displays the ratio of reviewed red IDP checks to total red IDP errors in the project.

The second row displays the ratio of reviewed checks to total checks having the color of the current check. In this example, this is the ratio of red errors reviewed to total red errors in the project. The third 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 upper-right part of the Viewer window.

```
example.c / Pointer_Arithmetic / line 104 / column 10
+ *p = 5; /* Out of bounds */
 NOW 
Error : pointer is outside its bounds
dereference of variable 'p' (pointer to int 32, size: 32 bits):
pointer is not null
points to 4 bytes at offset 400 in allocated buffer of 400 bytes
may point to variable or field of variable in: {Pointer_Arithmetic:array}
```

3 After you review the check, select an acronym to describe the check in the Predefined acronyms menu:

- **NOW** – Bug to fix now.
- **NXT** – Bug to fix in Next Release

- **ROB** – Robustness Issue
- **DEF** – Defensive Code
- **MIN** – Minor quality issue
- **OTH** – Other

Note You can also define your own acronyms. See “Defining Custom Acronyms”.

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

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

Coding review progress	Count	Progress
num IDP reviewed / num IDP to review (Red)	1/1	100
num reviewed / num to review (Red)	1/5	20
Software reliability indicator	113/230	49

Tracking Reviewed Checks in Procedural Entities View

The **Procedural entities** view in the Viewer displays which checks you have reviewed and the predefined acronym you used to describe each check.

Procedural entities	!	X	?	✓	Line	Col	%	Details	Reviewed	Acronym
Example_Project	5	44	9	43			91		<input type="checkbox"/>	
example.c	4	8	7	39	1		88	example.c	<input type="checkbox"/>	
+ Close_To_Zero ()			3	4	37	12	57	example.c	<input type="checkbox"/>	
+ Non_Infinite_Loop				5	88	11	100	example.c	<input type="checkbox"/>	
+ Pointer_Arithmetic	1	2	1	6	89	12	90	example.c	<input type="checkbox"/>	
✓ OVFL.2				1	94	23		operation [+] on scal...	<input type="checkbox"/>	
✓ IDP.3				1	96	6		pointer is within its b...	<input type="checkbox"/>	
✓ IRV.6				1	100	5		function returns an i...	<input type="checkbox"/>	
✓ IRV.7				1	102	9		function returns an i...	<input type="checkbox"/>	
! IDP.8	1				104	10		Error : pointer is out...	<input checked="" type="checkbox"/>	NOW
X UNR.10		1			107	8		unreachable code	<input type="checkbox"/>	
X OVFL.12		1			108	11		Unreachable check ...	<input type="checkbox"/>	
✓ IRV.13				1	112	6		function returns an i...	<input type="checkbox"/>	
? IDP.15			1		114	16		Warning : pointer m...	<input type="checkbox"/>	
✓ IDP.22				1	119	6		pointer is within its b...	<input type="checkbox"/>	

Tip If you do not see the Reviewed column, resize the **Procedural entities** view to display the column. If it does not appear, right click the **Procedural entities** column heading and select **Reviewed**.

You can select the **Reviewed** check box to mark a check as reviewed. Selecting this check box also automatically:

- Selects the check box for that check in the current check view (upper-right part of the window).
- Updates the counts in the coding review progress view (upper-left part of the window).

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 4-17

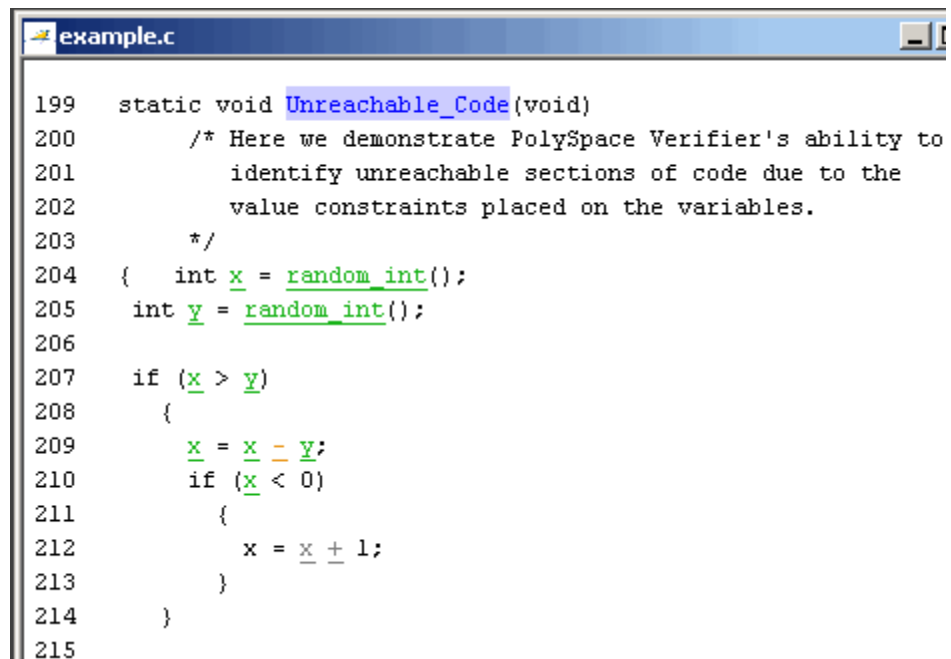
- “Example: Arithmetic Error” on page 4-18
- “Example: A Function with No Errors” on page 4-18
- “Example: Division by Zero” on page 4-19

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 for this function displays in the source code view.



```
199  static void Unreachable_Code(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  {  int x = random_int();
205     int y = random_int();
206
207     if (x > y)
208     {
209         x = x - y;
210         if (x < 0)
211         {
212             x = x + 1;
213         }
214     }
215
```

- 2 Examine the source code.

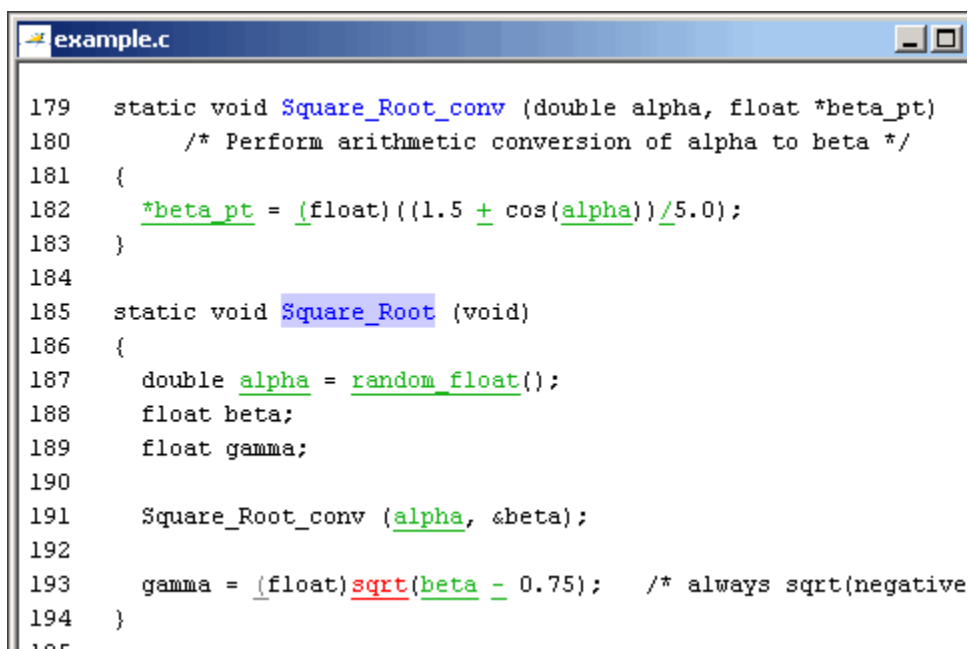
At line 212, the code `x = x + 1` is never reached because the condition `x < 0` is always false.

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 for this function displays in the source code view.



```
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;
190
191     Square_Root_conv (alpha, &beta);
192
193     gamma = (float)sqrt(beta - 0.75); /* always sqrt(negative
194 }
```

- 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 determines, in code with a large number of iterations, that a loop terminates and a variable does not overflow:

- 1 In **Procedural entities**, click the green `Non_Infinite_Loop()` function.

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

```
example.c
66  static int Non_Infinite_Loop (void)
67  {  const int big = 1073741821 ; /* 2**30-3 */
68  int x=0, y=0;
69
70  while (1)
71
72  {
73      if (y > big) { break;}
74      x = x + 2;
75      y = x / 2;
76  }
77
78  y = x / 100;
79  return y;
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 for this function displays in the source code view.

```
example.c
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)
152 {   int x=random_int();
153
154
155     if ((x>=4) && (x < -1))
156     {
157         Recursion( &x ); // always encounters a division by zero
158     }
159
160
161     x = 10;
162     if (random_int() > 0)
163     {
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 will result in division by zero. The orange color indicates that this 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 the Types of Checks That You See

You can filter the checks that you see in the Viewer so that you can focus on certain types of checks. PolySpace software provides three predefined composite filters, a custom composite filter, and several individual filters. You learn about filters in the following sections:

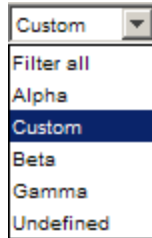
- “Using Composite Filters” on page 4-21
- “Using the Custom Filter” on page 4-23
- “Using Individual Filters” on page 4-25

Using Composite Filters

Composite filters combine individual filters, allowing you to display or hide groups of checks.

Use this filter...	To...
Alpha	Display all checks
Beta	Hide NIV, NIVL, NIP, Scalar OVFL, and Float OVFL checks
Gamma	Display red and gray checks
User def	Hide checks as defined in a custom filter that you can modify

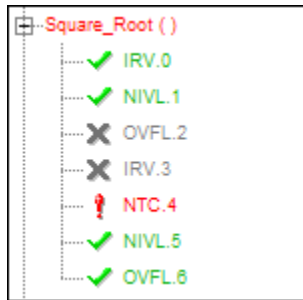
The default filter is `Custom`. You learn more about the `User def` filter in “Using the Custom Filter” on page 4-23. You select a composite filter from the filter menu.



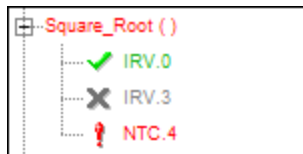
To learn how the composite filters affect the display of checks:

- 1 Select Alpha from the filter menu to display all the checks for `Square_Root()`.

`Square_Root()` has seven checks: four green, one red, and two gray.

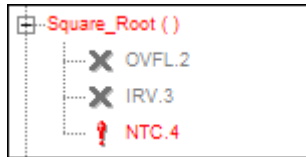


- 2 Select Beta from the filter menu to hide the NIV, NIVL, NIP, Scalar OVFL and Float OVFL checks.



Now, only three checks are visible: two IRV and one NTC.

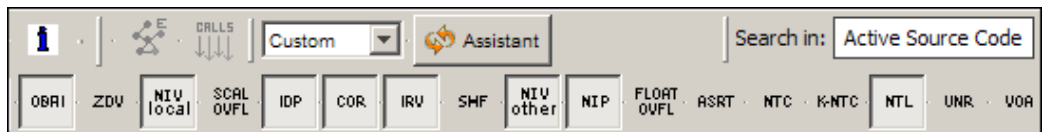
- 3 Select Alpha to display all checks again.
- 4 Select Gamma to display only the red and gray checks.



Now, only three checks are visible: one red and two gray.

Using the Custom Filter

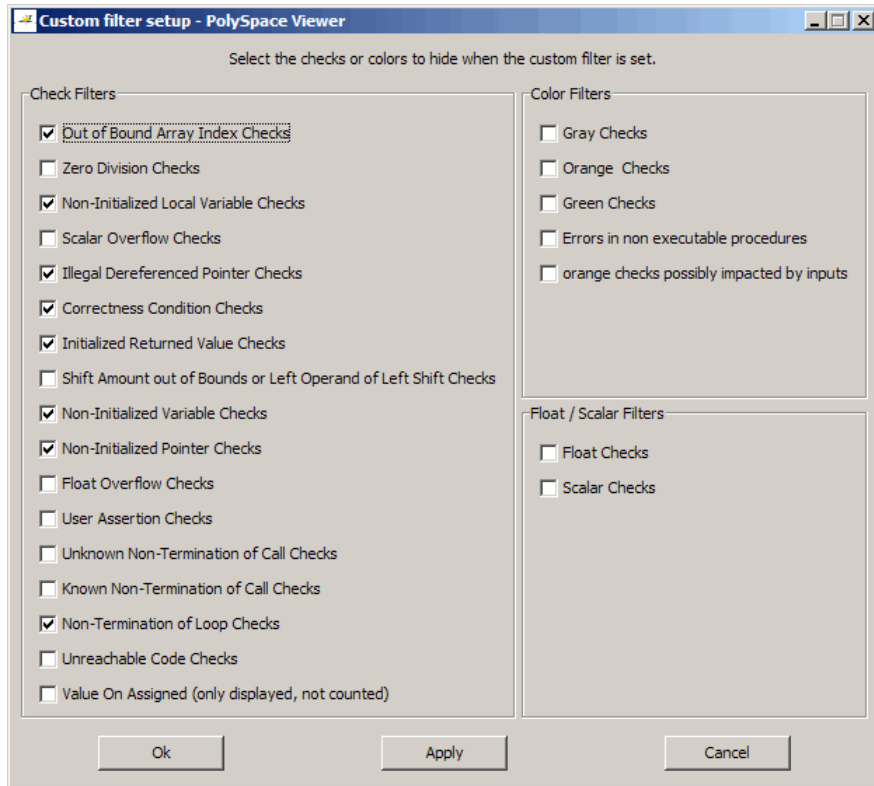
The custom filter is a composite filter that you define. It appears on the composite filter menu as `User def` and is the default composite filter. By default, the custom filter hides the OBAI, NIV local, IDP, COR, IRV, NIV other, NIP, and NTL checks as shown in the following figure.



To modify the custom filter:

- 1** Select `Custom` from the composite filters menu.
- 2** Select **Edit > Custom filters**.

The **Custom filter setup** dialog box appears.



- 3 Clear the filters for the checks that you want to display. For example, if you clear the **Out of Bound Array Index Checks** box, these checks display.

Note You do not have to change any of the selections for this tutorial.

- 4 Select the filters for the checks that you do not want to display.
- 5 Click **OK** to apply the changes and close the dialog box.

PolySpace software saves the custom filter definition in the Viewer preferences.

Using Individual Filters

You can use an individual filter to display or hide a given check category, such as VOA. When a filter is enabled, that check category does not display. For example, when the VOA filter is enabled, VOA checks do not display. When the filter is disabled, that check category displays. For example, when the VOA filter is disabled, VOA checks display. You can also filter by check color. To enable or disable an individual filter, click the toggle button for that filter on the toolbar.

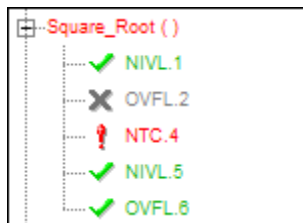
Tip The tooltip for a filter button tells you what filter the button is for and whether the filter is enabled or disabled.

To learn how an individual filter affects the display of checks:

- 1 Expand `Square_Root()`.
- 2 Select Alpha from the composite filters menu to display all checks.
- 3 Click the IRV filter button



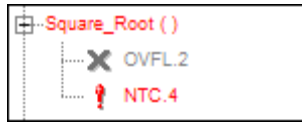
to hide the IRV check for `Square_Root()`.



- 4 Click the IRV filter button again to display the IRV check.
- 5 Click the green checks filter button



to hide the green checks.



Note When you filter a check category, some red checks with that category display. For example, if you filter IDP checks, IDP.8 still displays under Pointer_Arithmetic().

Reviewing Results in Assistant Mode

In this section...

- “What Is Assistant Mode?” on page 4-27
- “Switching to Assistant Mode” on page 4-27
- “Selecting the Methodology and Criterion Level” on page 4-28
- “Exploring Methodology for C” on page 4-28
- “Reviewing Checks” on page 4-30
- “Defining a Custom Methodology” on page 4-32

What Is 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 to you 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 selected methodology and criterion level

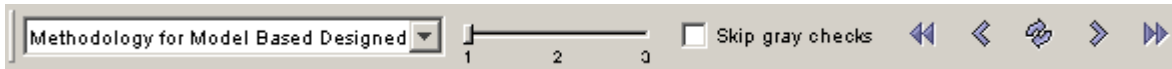
You learn about methodologies and criterion levels in “Selecting the Methodology and Criterion Level” on page 4-28.

Switching to Assistant Mode

If the Viewer is in assistant mode, the mode toggle button displays **Expert**. If the Viewer is in expert mode, the mode toggle button displays **Assistant**. To switch from expert mode to assistant mode:

- Click the Viewer mode button .

The Viewer window toolbar displays controls specific to assistant mode.



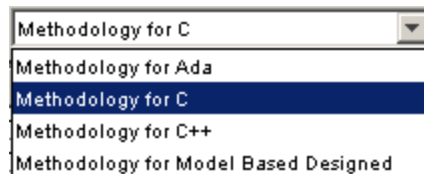
The controls for assistant mode include:

- A menu for selecting the review methodology for orange checks
- A slider for selecting the criterion level within that methodology
- A check box for skipping gray checks
- Arrows for navigating through the reviews

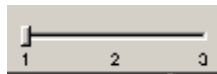
Selecting the Methodology and Criterion Level

A methodology is a named configuration that defines the number of orange checks, by category, that you review in assistant mode. Each methodology has three criterion levels. Each level specifies the number of orange checks for a given category. The levels correspond to different development phases that have different review requirements. 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 the configuration for **Methodology for C**. To examine the configuration for **Methodology for C**

- 1 Select **Edit > Preferences**.

The **Preferences PolySpace Viewer** dialog box appears.

2 Select the **Assistant configuration** tab.

The configuration for Methodology for C appears.

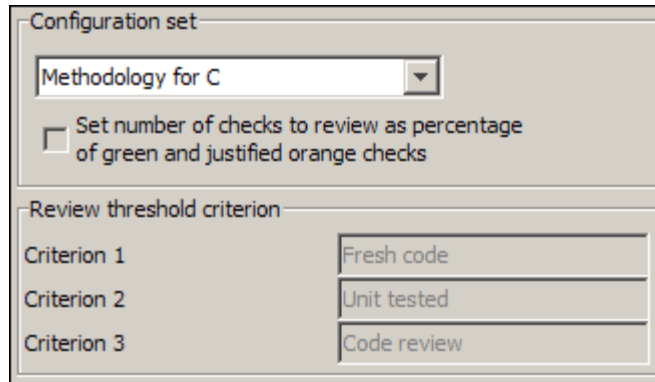
On the right side of the dialog box, a table shows the number of orange checks that you review for a given criterion and check category.

The screenshot shows the 'Assistant configuration' tab in the PolySpace Viewer. It features a table titled 'Number of checks to review' with columns for 'Criterion 1', 'Criterion 2', and 'Criterion 3'. The table is organized into three sections: 'Common', 'C & C++ only', and 'C only'. Each section lists check categories with their respective review counts for each criterion.

	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 Viewer toolbar (in assistant mode).



For the configuration Methodology for C, the criterion names are:

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:

- 1** All reds
- 2** All blocks of gray checks (the first check in each unreachable function)

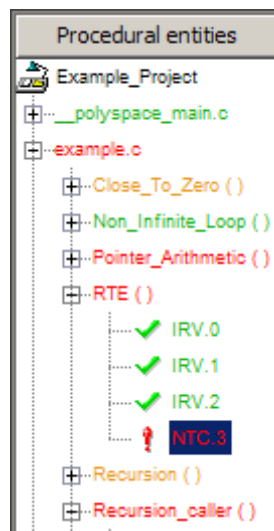
Note You can skip gray checks by selecting the **Skip gray checks** check box in the toolbar.

3 Orange checks according to the selected methodology and criterion level

Earlier in this tutorial, you selected Methodology for C, criterion 1. 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 .

In the procedural entities view (lower left), `RTE ()` expands and `NTC.3` is the current check.

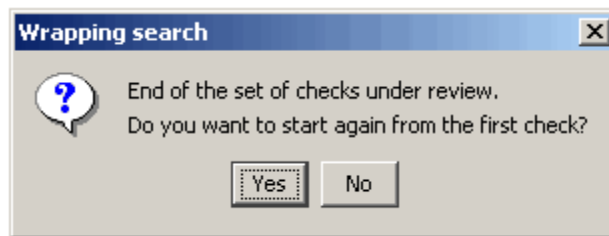


The source code view (lower right) displays the source for this check and the current check view (upper right) displays information about this check.

Note You can display the calling sequence and track review progress as you did in “Reviewing Results in Expert Mode” on page 4-10.

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

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



- 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 **Edit > Preferences**.

The **Preferences PolySpace Viewer** dialog box appears.

- 2 Select the **Assistant configuration** tab.
- 3 Select **Add a set** from the menu in **Configuration set**.
- 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**. This tells PolySpace software to select 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  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).

- 4** End PolySpace Viewer by selecting **File > Quit**.

Automatically Testing Unproven Code

Reviewing orange code to find true errors is time-consuming. 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 4-35
“Generating Report for example.c” on page 4-36

PolySpace Report Generator Overview

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

The PolySpace Report Generator provides the following report templates:

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

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. RTF format is used instead.

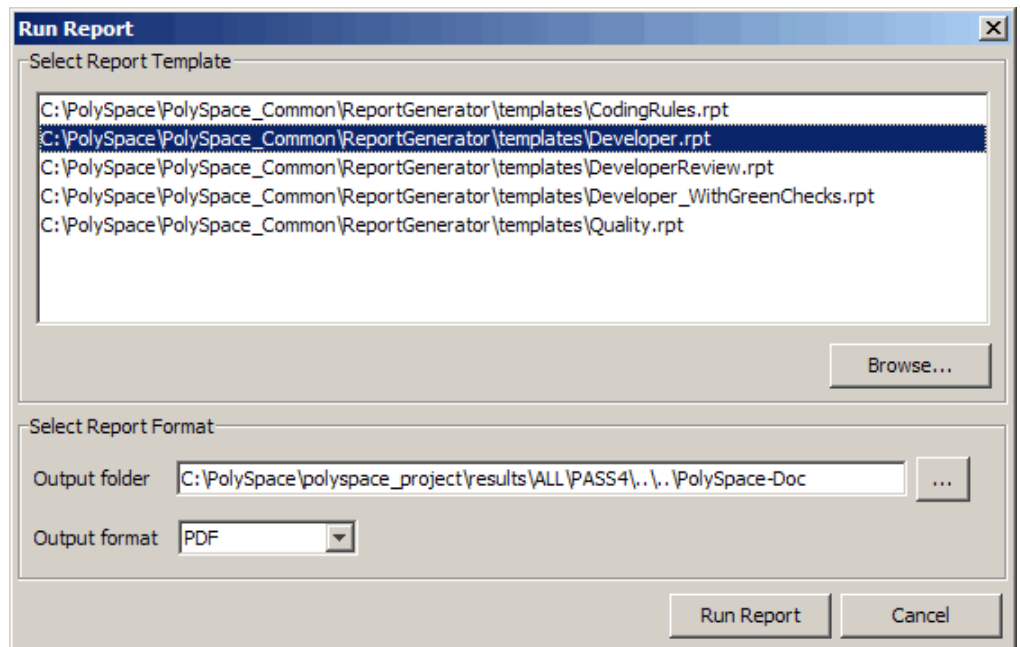
Generating Report for example.c

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

To generate a verification report:

- 1 In the Viewer, open your verification results if they are not already open.
- 2 Select **Reports > Run Report**.

The Run Report dialog box opens.



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

The software creates the specified report. When report generation completes, the report opens.

Checking MISRA C Compliance

- “About This Tutorial” on page 5-2
- “Setting Up MISRA C Checking” on page 5-3
- “Running a Verification with MISRA C Checking” on page 5-11

About This Tutorial

In this section...
“Overview” on page 5-2
“Before You Start” on page 5-2

Overview

PolySpace software can check that C code complies with MISRA C 2004 standards.³ To check MISRA C compliance, 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 MISRA C violations, you run the verification again.

In this tutorial, you learn how to:

- 1 Set an option for checking MISRA C compliance.
- 2 Select MISRA C rules to check.
- 3 Run a verification with MISRA C checking.

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 Project File”.

3. 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 the Example Project” on page 5-3

“Setting the MISRA C Checking Option” on page 5-4

“Creating a MISRA C Rules File” on page 5-4

“Excluding Files from the MISRA C Checking” on page 5-8

“Configuring Text and XML Editors” on page 5-8

“Saving the Project with a New Name” on page 5-10

Opening the Example Project

For this tutorial, you modify the project in `example.cfg` to include MISRA C checking and save the project with a new name.⁴ You use the Launcher to modify the project.

To open the Launcher:

- Double-click the Launcher icon.

To open `example.cfg`:

- 1** Select **File > Open project**.

The **Please select a file** dialog box opens.

- 2** In **Look in**, navigate to `polyspace_project`.

- 3** Select `example.cfg`.

- 4** Click **Open** to open the file and close the dialog box.

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

Setting the MISRA C Checking Option

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

- 1 In the analysis options part of the Launcher window, expand the **Compliance with standards** option.
- 2 Select the **Check MISRA-C:2004 rules** check box.
- 3 Expand the **Check MISRA-C:2004 rules** option.

Two options, **Rules configuration** and **Files and folders to ignore**, appear.

<input type="checkbox"/> Check MISRA C rules	<input type="checkbox"/>		
MISRA C rules configuration		...	-misra2
Files and folders to ignore		...	-includes-to-ignore

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


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 5-4
- “Setting All the Rules to Off” on page 5-5
- “Selecting the Rules to Check” on page 5-5

Opening a New Rules File

To open a new rules file:

- 1 Click the button  to the right of the **Rules configuration** option.

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

2 Select File > New File.

A table of rules appears. For each rule, you specify one of these states:

State	Causes the verification to...
Error	End after the compile phase when this rule is violated.
Warning	Display warning message and continue verification when this rule 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 state **Error** (you cannot change the state of these).

Setting All the Rules to Off

Because this tutorial checks only a few rules, first set the state of all rules to **Off**. Later, you select the rules 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 names **17 Pointers and Arrays**.

4 Select the **Warning** column for 17.4.

The completed rules table looks like:

Rules	Error	Warning	Off
MISRA C rules			
— Number of rules by mode :	7	1	134
+1 Environment			
+2 Language extensions			
+3 Documentation			
+4 Character sets			
+5 Identifiers			
+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			
+15 Switch statements			
-16 Functions			
—16.1 Functions shall not be defined with variable numbers of arguments.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
—16.2 Functions shall not call themselves, either directly or indirectly.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
—16.3 Identifiers shall be given for all of the parameters in a function prototype.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
—16.4 The identifiers used in the declaration and definition of a function shall match.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
—16.5 Functions with no parameters shall be declared with parameter type void.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
—16.6 The number of arguments passed to a function shall match the number in the function prototype.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
—16.7 A pointer parameter in a function prototype should be declared as pointer to void.	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
—16.8 All exit paths from a function with non-void return type shall have an explicit return statement.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
—16.9 A function identifier shall only be used with either a preceding &, or with a preceding *.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
—16.10 If a function returns error information, then that error information shall be returned via a pointer to a structure.	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
-17 Pointer and arrays			
—17.1 Pointer arithmetic shall only be applied to pointers that address an array of the same type.	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
—17.2 Pointer subtraction shall only be applied to pointers that address elements of the same type.	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
—17.3 >, >=, <, <= shall not be applied to pointer types except where they point to the same object.	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
—17.4 Array indexing shall be the only allowed form of pointer arithmetic.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
—17.5 The declaration of objects should contain no more than 2 levels of pointer indirection.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
—17.6 The address of an object with automatic storage shall not be assigned to a pointer with static storage.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
+18 Structures and unions			
+19 Preprocessing directives			
+20 Standard libraries			
+21 Run-time failures			

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  to the right of the **Files and folders to ignore** option.

2 Click the folder icon.



The **Select a file or folder to include** dialog box appears.

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® rules, you should configure your text and XML editors in the Launcher. Configuring text and XML editors in the Launcher allows you to view source files and MISRA reports directly from the MISRA-C log in the Launcher.

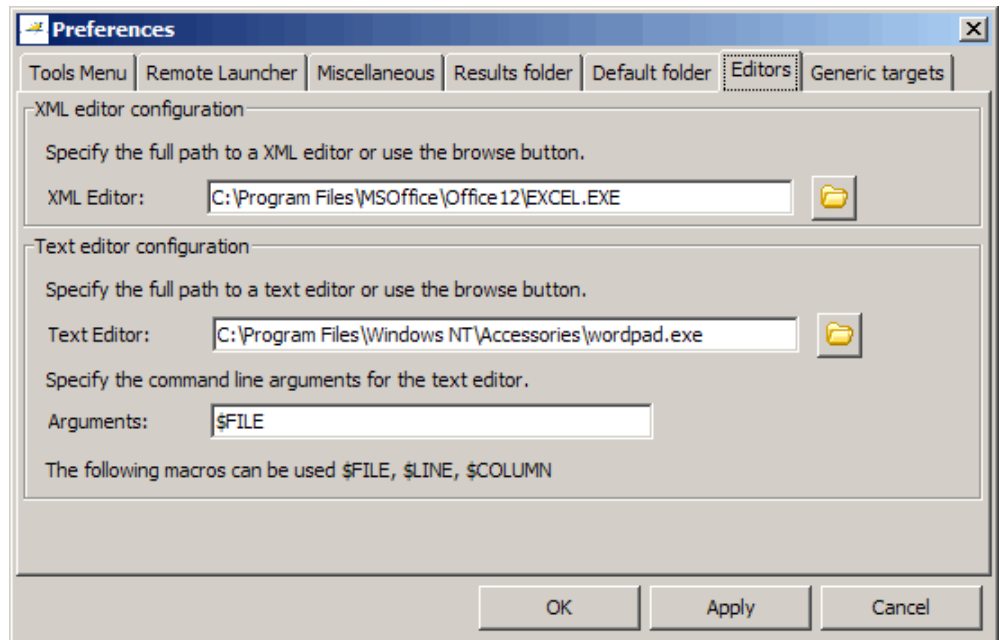
To configure your text and .XML editors:

1 Select **Edit > Preferences**.

The Preferences dialog box opens.

- 2 Select the **Editors** tab.

The Editors tab opens.



- 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 Launcher logs. For example:

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

- 5 Specify command line arguments for the text editor. For example:

\$FILE

- 6 Click **OK**.

Saving the Project with a New Name

You save the project with a new name so that you do not modify `example.cfg`. To save the project with the name `misra_example.cfg`:

- 1** Select **File > Save as new project**.
- 2** In the **Save the project as** dialog box, navigate to `polyspace_project`.
- 3** Enter `misra_example` for the **Session identifier** and `*.cfg` for the type.
- 4** Click **OK** to close the dialog box.

Running a Verification with MISRA C Checking

In this section...

- “Starting the Verification” on page 5-11
- “Examining the MISRA C Log” on page 5-12
- “Opening MISRA-C Report” on page 5-15

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.⁵

Note Some rules address run-time errors.

The verification stops if there is a violation of a rule with state Error.

To start the verification:

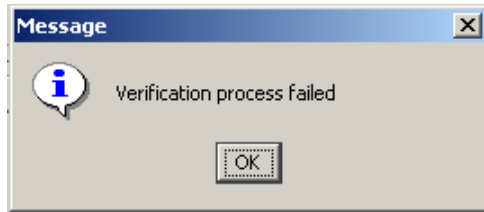
- 1 Click the **Start** button



- 2 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 verification fails because of MISRA C violations. A message dialog box appears.

5. MISRA and MISRA C are registered trademarks of MISRA Ltd., held on behalf of the MISRA Consortium.



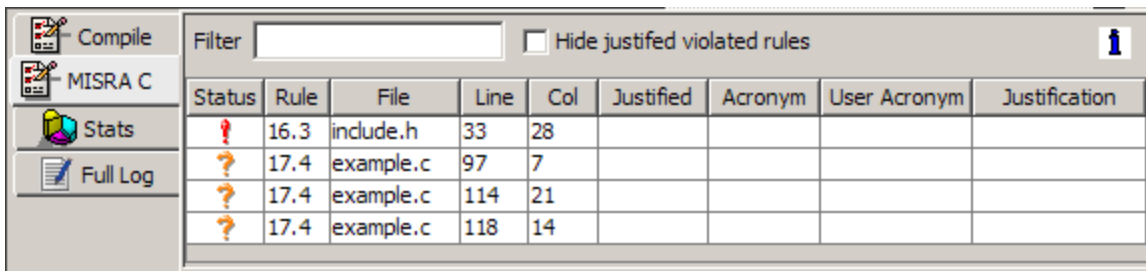
3 Click **OK**.

Examining the MISRA C Log

To examine the MISRA C violations:

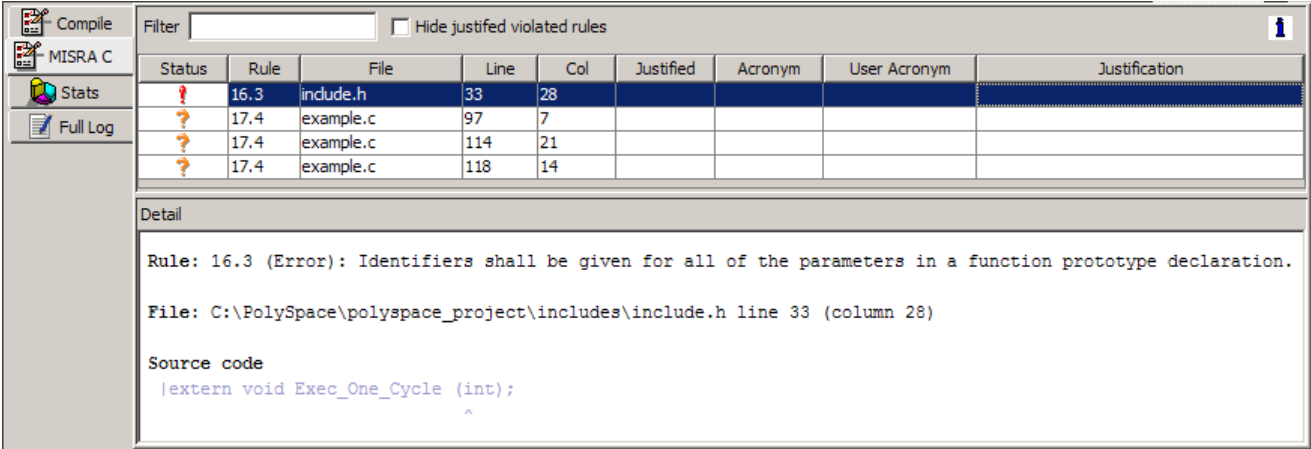
1 Click the **MISRA-C** button in the log area of the Launcher window.

A list of MISRA C violations appear in the log part of the window.

A screenshot of a software interface showing the MISRA C log. On the left is a sidebar with buttons for "Compile", "MISRA C", "Stats", and "Full Log". The main area has a "Filter" input field, a checkbox for "Hide justified violated rules", and an information icon. Below this is a table with columns: Status, Rule, File, Line, Col, Justified, Acronym, User Acronym, and Justification. The table contains four rows of data.

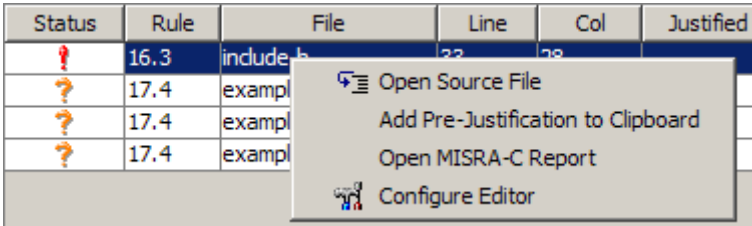
Status	Rule	File	Line	Col	Justified	Acronym	User Acronym	Justification
!	16.3	include.h	33	28				
?	17.4	example.c	97	7				
?	17.4	example.c	114	21				
?	17.4	example.c	118	14				

2 Click on 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.



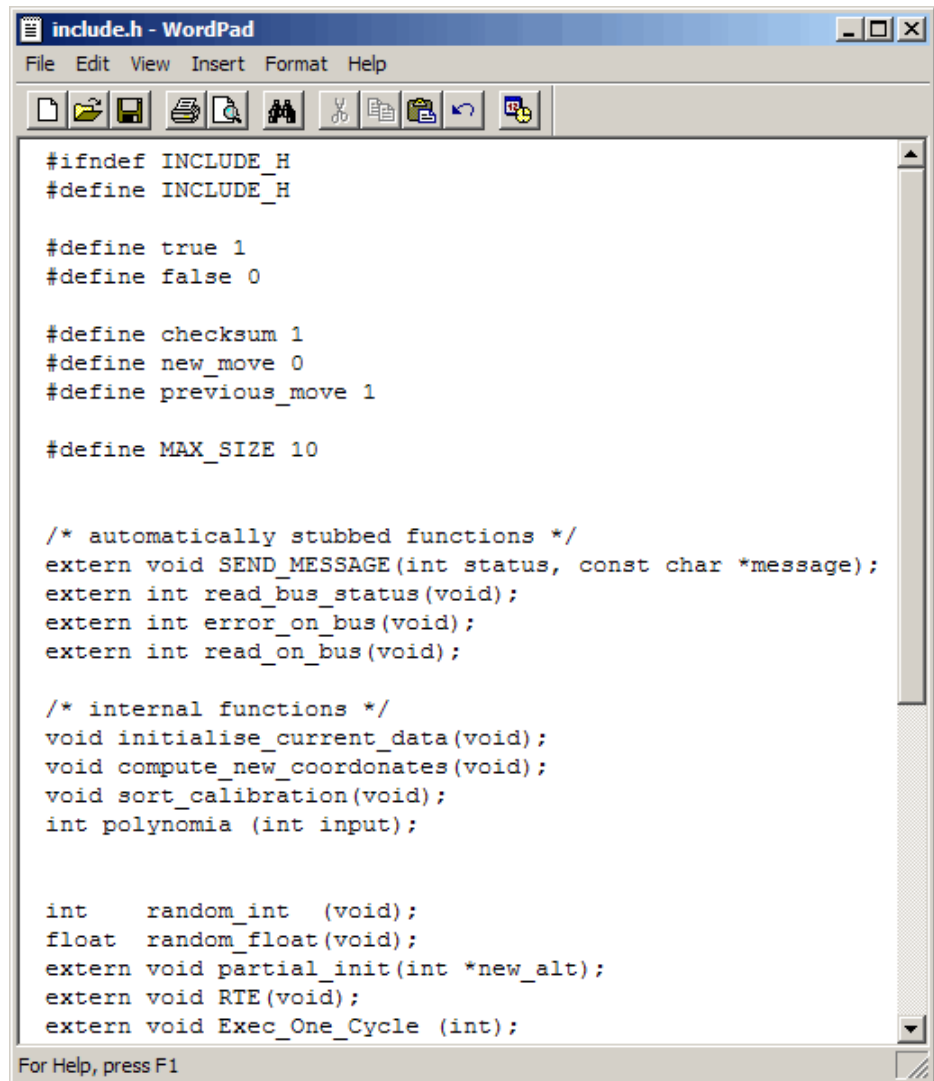
The log reports a violation of rule 16.3. A function prototype declaration in include.h is missing an identifier.

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



The include.h file opens in your text editor.

Note You must configure a text editor before you can open source files. See “Configuring Text and XML Editors” on page 5-8.



```
#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);

int random_int (void);
float random_float(void);
extern void partial_init(int *new_alt);
extern void RTE(void);
extern void Exec_One_Cycle (int);
```

- 4 Correct the MISRA violation and run the verification again.

The verification will complete, 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 You must configure an XML editor before you can open a MISRA-C report. See “Configuring Text and XML Editors” on page 5-8.

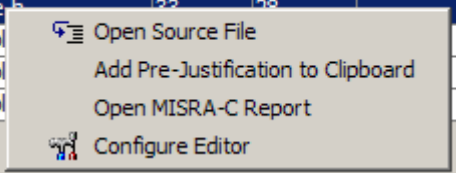
To view the MISRA-C report:

- 1 Click the **MISRA-C** button in the log area of the Launcher window.

A list of MISRA C violations appear in the log part of the window.

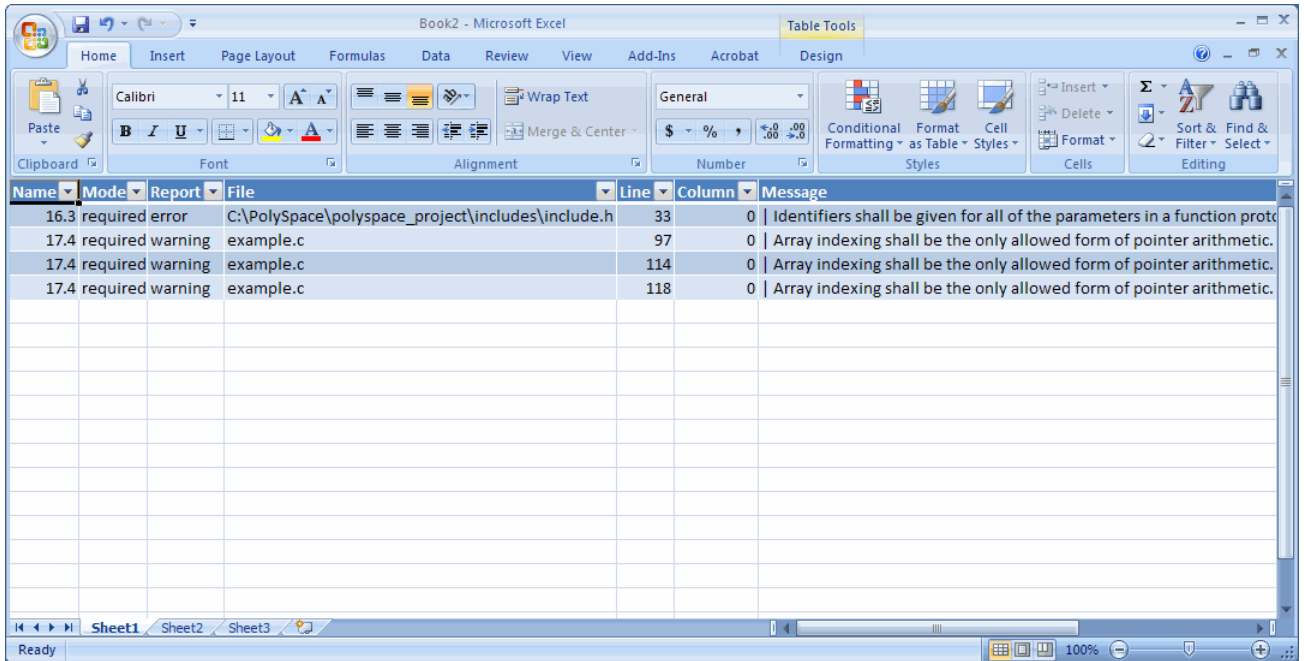
- 2 Right click any row in the log, and select **Open MISRA-C Report**.

Status	Rule	File	Line	Col	Justified
!	16.3	include.h	22	23	
?	17.4	exampl			
?	17.4	exampl			
?	17.4	exampl			



The report opens in your XML editor.

5 Checking MISRA C® Compliance



The screenshot shows a Microsoft Excel spreadsheet titled "Book2 - Microsoft Excel" with the "Table Tools" ribbon selected. The spreadsheet contains a table with the following data:

Name	Mode	Report	file	Line	Column	Message
16.3	required	error	C:\PolySpace\polyspace_project\includes\include.h	33	0	Identifiers shall be given for all of the parameters in a function proto
17.4	required	warning	example.c	97	0	Array indexing shall be the only allowed form of pointer arithmetic.
17.4	required	warning	example.c	114	0	Array indexing shall be the only allowed form of pointer arithmetic.
17.4	required	warning	example.c	118	0	Array indexing shall be the only allowed form of pointer arithmetic.

Using a PolySpace Project Model File

- “About This Tutorial” on page 6-2
- “Creating a New PolySpace Project Model File” on page 6-3
- “Creating a Configuration File from a PolySpace Project Model File” on page 6-10
- “Deleting a Generic Target from the Preferences” on page 6-13

About This Tutorial

In this section...
“Overview” on page 6-2
“Before You Start” on page 6-2

Overview

A PolySpace project model file provides a way to save generic targets with project information. Although you can populate a project with information, such as source files and project options, from a project model file, you cannot run a verification with a project model file. You must have a configuration file to run a verification. In this tutorial, you learn how to:

- 1** Create a new project model file.
- 2** Define a generic target and save it in the project model file.
- 3** Create a configuration file from a project model file.
- 4** Delete a generic target from the Launcher preferences.

Before You Start

Before you start this tutorial, you must complete Chapter 2, “Setting Up a Project File” to learn about configuration files and basic Launcher operations.

Creating a New PolySpace Project Model File

In this section...

“What Is a PolySpace Project Model File?” on page 6-3

“Creating the PolySpace Project Model File” on page 6-3

What Is a PolySpace Project Model File?

A PolySpace project model file is a project file that includes generic target processors. A development team uses this file to share project information. The workflow is:

- 1** A team leader creates a project model file (.ppm). This file has the analysis options for the project, including generic targets.
- 2** The team leader distributes the .ppm file to the team.
- 3** A developer opens the .ppm file. From this file, PolySpace software populates the project parameters and the generic targets in the preferences.
- 4** The developer adds source files, include folders, and a results folder to the project and saves it as a configuration file (.cfg).
- 5** The developer launches a verification with the .cfg file.

Creating the PolySpace Project Model File

You use the PolySpace Launcher to create a PolySpace project model file. Creating a project model file involves:

- “Opening a New Project” on page 6-4
- “Examining the Preferences Before Adding the Generic Target” on page 6-4
- “Defining the Generic Target” on page 6-5
- “Examining the Preferences After Adding the Generic Target” on page 6-7
- “Saving the PolySpace Project Model File” on page 6-8

Opening a New Project

To open a new project:

- 1 Open the PolySpace Launcher by double-clicking the Launcher icon on your desktop.
- 2 If the **PolySpace Language Selection** dialog box appears, select **PolySpace for C/C++** and click **OK**.
- 3 Select **File > New Project**.
- 4 In the **Choose the language** dialog box, select **C** and click **OK** to close the dialog box.

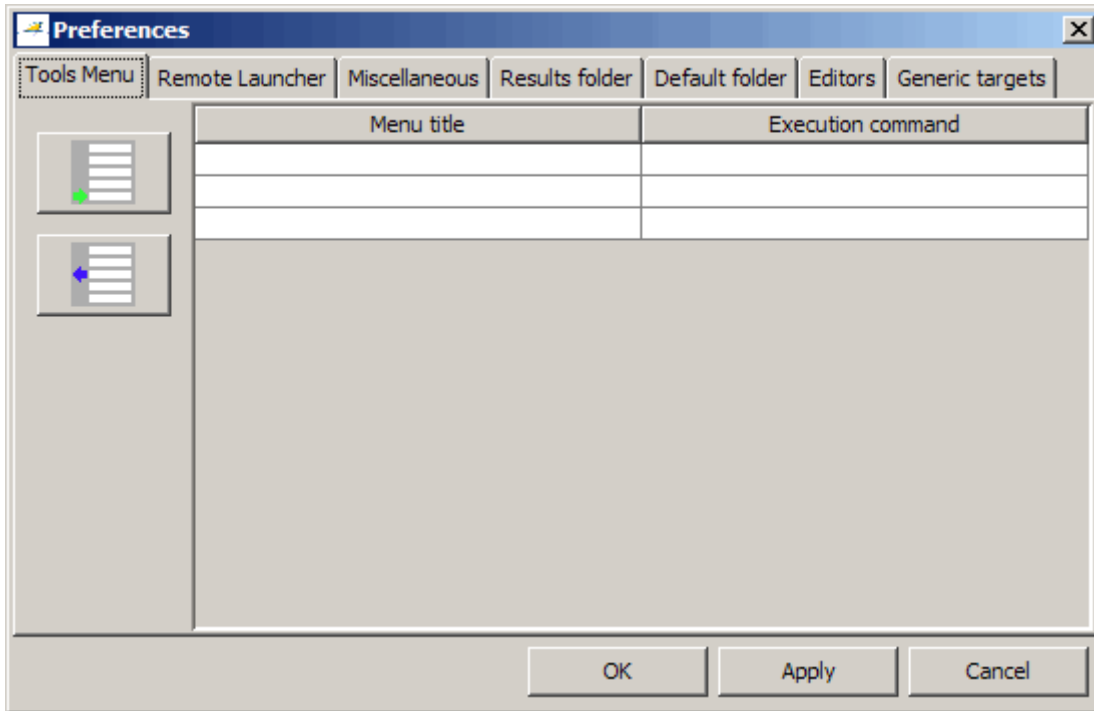
Examining the Preferences Before Adding the Generic Target

In this step, you look at the generic targets in the preferences before you add a generic target. Unless you previously added a generic target, the generic targets list is empty. Later, after you add a generic target, you look at the generic targets in the preferences again, and see that the generic target that you added is in the list.

To look at the generic targets in the preferences:

- 1 Select **Edit > Preferences**.

The **Preferences** dialog box appears.



- 2 Select the **Generic targets** tab.

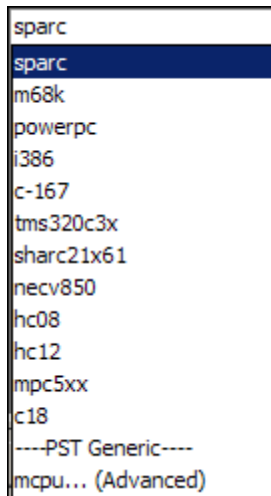
Unless you previously added generic targets to your preferences, the generic targets list is empty.

- 3 Click **Cancel** to close the dialog box.

Defining the Generic Target

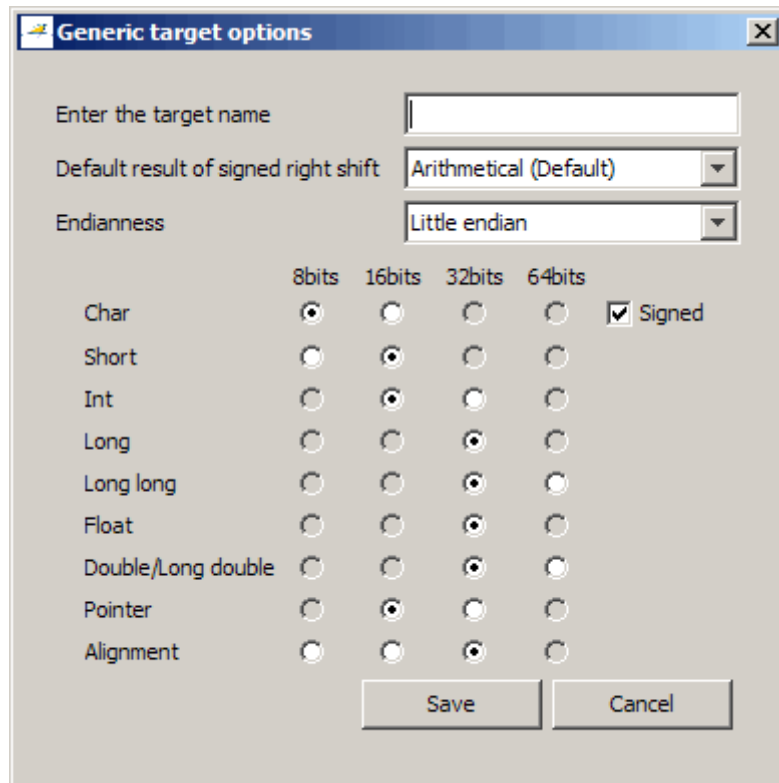
To define a generic target:

- 1 In **Analysis options**, expand **Target/Compilation**.
- 2 Click the down arrow to open the **Target processor type** menu.



3 Select **mcpu... (Advanced)**.

The **Generic target options** dialog box appears.



4 In **Enter the target name**, enter target1.

5 Click **Save** to save the generic target options and close the dialog box.

Examining the Preferences After Adding the Generic Target

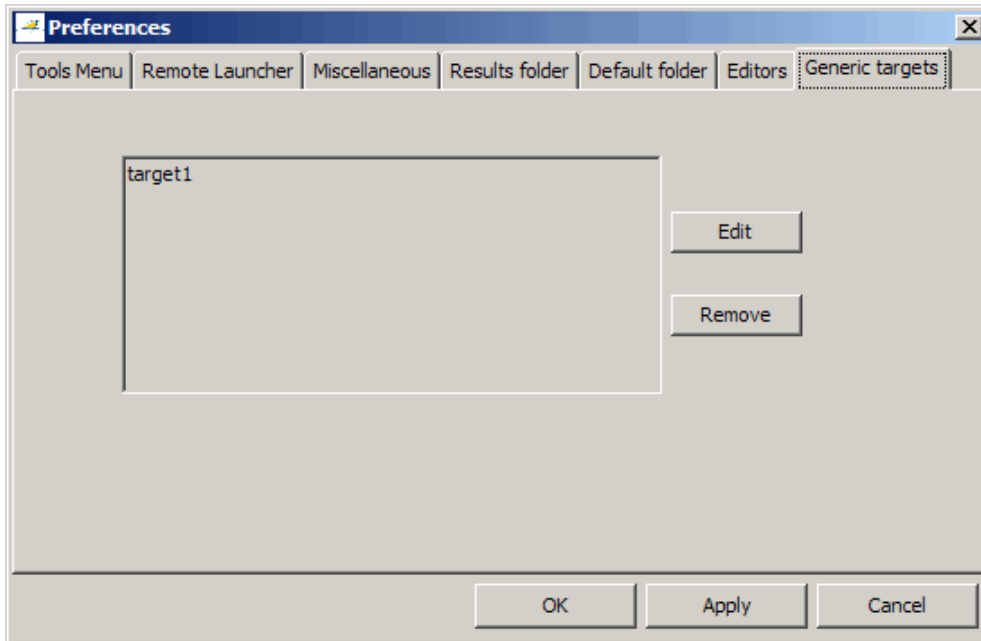
Now when you look at the generic targets in the preferences, you should see the generic target that you added. To look at the generic targets list in the preferences:

1 Select **Edit > Preferences**.

The **Preferences** dialog box appears.

2 Select the **Generic targets** tab.

Notice that `target1` appears in the generic targets list:



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

Saving the PolySpace Project Model File

To save the PolySpace project model file:

1 Select **File > Save project**.

The **Save the project as** dialog box appears.

2 Select ***.ppm** from the **Files of type** menu.

3 In **Session identifier**, enter `target_example`.

4 Click **OK** to save the file and close the dialog box.

Note The generic target that you defined in this tutorial remains in your preferences until you delete it. Be sure to complete the section “Deleting a Generic Target from the Preferences” on page 6-13 at the end of this tutorial.

Creating a Configuration File from a PolySpace Project Model File

In this section...

- “Why You Must Have a Configuration File” on page 6-10
- “Opening the Project Model File” on page 6-10
- “Entering Additional Required Information” on page 6-11
- “Saving the Configuration File” on page 6-11

Why You Must Have a Configuration File

In the first part of this tutorial, you created a project model file. To run a verification, you must have a configuration file. In this part of the tutorial, you create a configuration file from the project model file that you created earlier. The workflow is:

- 1 Open the project model file. Opening the project model file populates the:
 - Generic targets in the preferences
 - Analysis options and other project information
- 2 Enter additional information, such as the results folder and source files.

Note If you enter the results folder and source files in the project before you save it as a PolySpace project model file, then that information is saved in the file and appears in the project when you open the file.

- 3 Save the configuration file.

Opening the Project Model File

To open the project model file:

- 1 Select **File > Open project**.

The **Please select a file** dialog box appears.

- 2 Navigate to the `polyspace_project` folder.
- 3 In **File of type:**, select **Project Model (*.ppm)** files from the menu.
- 4 Select `target_example.ppm` and click **Open**.

A message appears telling you that this project has no source files.
- 5 Click **OK** to close the message dialog box.

Entering Additional Required Information

A configuration file must specify the source files and results folder.

To complete the required project information:

- In **Results folder**, enter the results folder that you created. For the example in this guide, it is `C:\polyspace_project\results`.
- Add `C:\polyspace_project\sources\example.c` to the source files.
- Add `C:\polyspace_project\includes` to the include folders.

Note For more information about adding source files and include folders to a project, see “Creating a New Project to Verify the Example C File” on page 2-9.

Saving the Configuration File

To save the configuration file:

- 1 Select **File > Save project**.

The **Save the project as** dialog box appears.
- 2 Navigate to the `polyspace_project` folder.
- 3 In **Session identifier**, enter `example2`.
- 4 Leave the default type as `*.cfg`.
- 5 Click **OK** to save the project and close the dialog box.

Note Your preferences still include the generic target `target1` . Complete “Deleting a Generic Target from the Preferences” on page 6-13 to delete this generic target from your preferences.

Deleting a Generic Target from the Preferences

In this section...

“Understanding the Generic Targets Preference” on page 6-13

“Deleting the Generic Target Added in This Tutorial” on page 6-13

Understanding the Generic Targets Preference

The list of generic targets is stored as a PolySpace software preference. You can add generic targets to the list in one of these ways:

- Edit the preferences using the PolySpace Launcher.
- Open a PolySpace project model file that includes generic targets.

The generic targets remain in your preferences until you delete them. You should delete the generic target that you defined and added to your preferences earlier in this tutorial.

Deleting the Generic Target Added in This Tutorial

To delete the generic target `target1` from your preferences:

- 1** In **Analysis options**, expand **Target/Compilation**.
- 2** If **Target processor type** is `target1`, change it to `sparc` (You cannot delete a generic target if it is the target processor type for the current project.)
- 3** Select **Edit > Preferences**.
The **Preferences** dialog box appears.
- 4** Select the **Generic targets** tab.
- 5** Select `target1` from the list.
- 6** Click **Remove**.
- 7** Click **OK** to apply the change and close the dialog box.

Note You removed the generic target `target1` from your preferences, but it is still in `target_example.ppm`. If you save the current project in `target_example.ppm`, then `target_example.ppm` will no longer include `target1`.

A

- active project
 - definition 3-15
 - setting 3-15
- analysis options 2-14
 - generic targets 6-5
 - MISRA C compliance 5-4
- ANSI compliance 3-5
- AOT. *See* Automatic Orange Tester
- assistant mode
 - criterion 4-28
 - custom methodology 4-32
 - methodology 4-28
 - methodology for C 4-28
 - overview 4-27
 - reviewing checks 4-30
 - selection 4-27
 - use 4-27 4-30
- Automatic Orange Tester
 - overview 4-34

C

- call graph 4-12
- call tree view 4-5
- calling sequence 4-12
- cfg. *See* configuration file
- client 1-5 3-2
 - installation 1-6
 - verification on 3-24
- coding review progress view 4-5 4-13
- color-coding of verification results 1-2 4-7
- compile log
 - Launcher 3-25
 - Spooler 3-7
- compile phase 3-5
- compliance
 - ANSI 3-5
 - MISRA C 1-2 5-1
- composite filters 4-21

- configuration file
 - definition 2-3
- custom methodology
 - definition 4-32

D

- default folder
 - changing in preferences 2-7
- desktop file
 - definition 2-3
- division by zero
 - example 4-19
- downloading
 - results 3-10
- dsk. *See* desktop file

E

- expert mode
 - filters 4-21
 - composite 4-21
 - individual 4-25
 - overview 4-10
 - selection 4-10
 - use 4-10

F

- files
 - includes 2-11
 - results 2-11
 - source 2-11
- filters 4-21
 - alpha 4-21
 - beta 4-21
 - custom
 - modification 4-23
 - use 4-23
 - gamma 4-21
 - individual 4-25

- user def 4-21
- folders
 - includes 2-11
 - results 2-11
 - sources 2-11

G

- generic target processors
 - adding 6-4
 - definition 6-5
 - deleting 6-13

H

- hardware requirements 3-12
- help
 - accessing 1-10

I

- installation
 - PolySpace Client for C/C++ 1-6
 - PolySpace products 1-6
 - PolySpace Server for C/C++ 1-6

L

- Launcher 1-5
 - monitoring verification progress 3-25
 - opening 2-5
 - starting verification on client 3-24
 - starting verification on server 3-5
 - stopping 3-26
 - viewing logs 3-25
 - window 2-5
 - overview 2-5
 - progress bar 3-25
- licenses
 - obtaining 1-6
- logs

- compile
 - Launcher 3-25
 - Spooler 3-7
- full
 - Launcher 3-25
 - Spooler 3-7
- stats
 - Launcher 3-25
 - Spooler 3-7
- viewing
 - Launcher 3-25
 - Spooler 3-7

M

- methodology for C 4-28
- MISRA C compliance 1-2
 - analysis option 5-4
 - checking 5-1
 - file exclusion 5-8
 - log 5-12
 - rules file 5-4

P

- PolySpace Client for C/C++
 - installation 1-6
 - license 1-6
- PolySpace In One Click
 - active project 3-15
 - overview 3-15
 - sending files to PolySpace software 3-17
 - starting verification 3-17
 - use 3-15
- PolySpace products for C
 - components 1-5
 - installation 1-6
 - licenses 1-6
 - overview 1-2
 - related products 1-11

- user interface 1-5
 - workflow 1-7
- PolySpace project model file
 - creation 6-3
 - definition 6-3
 - overview 6-2
 - use 6-1
- PolySpace Queue Manager Interface. *See* Spooler
- PolySpace Server for C/C++
 - installation 1-6
 - license 1-6
- ppm. *See* PolySpace project model file
- preferences
 - Launcher
 - default folder 2-7
 - default server mode 3-5
 - generic targets 6-4
 - server detection 3-13
 - Viewer
 - assistant configuration 4-28
- procedural entities view 4-5
 - reviewed column 4-15
- product overview 1-2
- progress bar
 - Launcher window 3-25
- project
 - creation 2-3 2-9
 - 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 2-16
- project model file. *See* PolySpace project model file

R

- related products 1-11
 - PolySpace products for linking to Models 1-11
 - PolySpace products for verifying Ada code 1-11
 - PolySpace products for verifying C++ code 1-11
- reports
 - generation 4-35
- results
 - downloading from server 3-10
 - folder 2-11
 - opening 4-4
 - report generation 4-35
 - reviewing 4-1
- reviewed column 4-15
- rte view. *See* procedural entities view

S

- selected check view 4-5
- server 1-5 3-2
 - detection 3-13
 - information in preferences 3-13
 - installation 1-6 3-13
 - verification on 3-5
- source code view 4-5
- Spooler 1-5
 - monitoring verification progress 3-7
 - removing verification from queue 3-10
 - use 3-7
 - viewing log 3-7

T

- troubleshooting failed verification 3-12

U

unreachable code
 example 4-17

V

variables view 4-5
verification
 Ada code 1-11
 C code 1-2
 C++ code 1-11
 client 3-2
 compile phase 3-5
 failed 3-12
 monitoring progress
 Launcher 3-25
 Spooler 3-7
 phases 3-5
 results
 color-coding 1-2
 opening 4-4
 report generation 4-35
 reviewing 4-1
 running 3-2
 running on client 3-24

 running on server 3-5
 starting
 from Launcher 3-2 3-5 3-24
 from PolySpace In One Click 3-2 3-17
 stopping 3-27
 troubleshooting 3-12
 with MISRA C checking 5-11

Viewer 1-5

 modes 4-3
 selection 4-3
 opening 4-3
 window
 call tree view 4-5
 coding review progress view 4-5
 overview 4-5
 procedural entities view 4-5
 selected check view 4-5
 source code view 4-5
 variables view 4-5

W

workflow
 basic 1-7
 in this guide 1-8