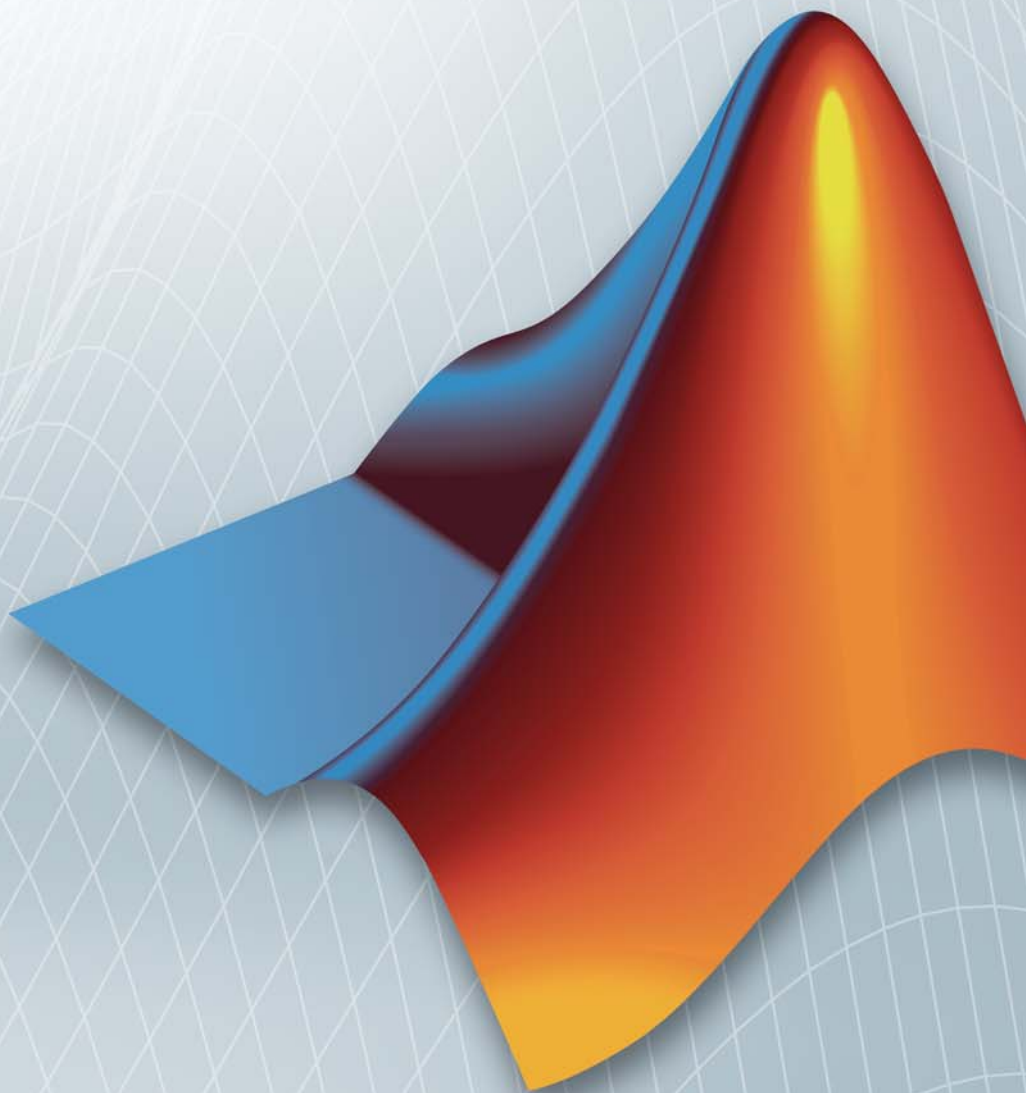


# Polyspace® Products for C++ 8

## Getting Started Guide



## How to Contact MathWorks



[www.mathworks.com](http://www.mathworks.com) Web  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab) Newsgroup  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html) Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com) Product enhancement suggestions  
[bugs@mathworks.com](mailto:bugs@mathworks.com) Bug reports  
[doc@mathworks.com](mailto:doc@mathworks.com) Documentation error reports  
[service@mathworks.com](mailto:service@mathworks.com) Order status, license renewals, passcodes  
[info@mathworks.com](mailto: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 1997–2011 by The MathWorks, Inc.

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

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

### Trademarks

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

### Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

**Revision History**

March 2008	First printing	Revised for Version 5.1 (Release 2008a)
October 2008	Second printing	Revised for Version 6.0 (Release 2008b)
March 2009	Third printing	Revised for Version 7.0 (Release 2009a)
September 2009	Online only	Revised for Version 7.1 (Release 2009b)
March 2010	Online only	Revised for Version 7.2 (Release 2010a)
September 2010	Fourth printing	Revised for Version 8.0 (Release 2010b)
April 2011	Online Only	Revised for Version 8.1 (Release 2011a)



## Introduction to Polyspace Products for Verifying C++ Code

1

<b>Product Overview</b> .....	1-2
Overview of Polyspace Verification .....	1-2
The Value of Polyspace Verification .....	1-2
<b>Product Components</b> .....	1-5
Polyspace Products for C++ .....	1-5
Polyspace Verification Environment .....	1-5
Other Polyspace Components .....	1-10
<b>Installing Polyspace Products</b> .....	1-12
Finding the Installation Instructions .....	1-12
Obtaining Licenses for Polyspace® Client for C/C++ and Polyspace® Server for C/C++ Products .....	1-12
<b>Working with Polyspace Software</b> .....	1-13
Basic Workflow .....	1-13
Tutorials in This Guide .....	1-14
<b>Additional Information and Support</b> .....	1-16
Product Help .....	1-16
MathWorks Online .....	1-16
<b>Related Products</b> .....	1-17
Polyspace Products for Verifying C Code .....	1-17
Polyspace Products for Verifying Ada Code .....	1-17
Polyspace Products for Linking to Models .....	1-17

## Setting Up a Polyspace Project

### 2

<b>About Setting Up a Project Tutorial</b> .....	2-2
Overview .....	2-2
Example Files .....	2-2
<b>Creating a New Project</b> .....	2-3
What Is a Project? .....	2-3
Preparing the Project Folders .....	2-4
Opening Polyspace Verification Environment .....	2-5
Creating a New Project to Verify a Class in the Training C++ File .....	2-7

## Running a Verification

### 3

<b>About Running a Verification Tutorial</b> .....	3-2
Overview .....	3-2
Before You Start .....	3-3
<b>Preparing for Verification</b> .....	3-4
Opening the Project .....	3-4
Specifying Source Files to Verify .....	3-4
Checking for Compilation Problems .....	3-5
<b>Launching Server Verification from Project     Manager</b> .....	3-10
Starting the Verification .....	3-10
Monitoring Progress of the Verification .....	3-12
Removing Verification Results from the Server .....	3-18
Troubleshooting a Failed Verification .....	3-19
<b>Using Polyspace In One Click to Launch Verification</b> ..	3-21
Overview of Polyspace In One Click .....	3-21
Setting the Active Project .....	3-21
Sending the Files to Polyspace Software .....	3-23

<b>Launching Client Verification from Project Manager</b> .....	<b>3-28</b>
Starting the Verification .....	<b>3-28</b>
Monitoring the Progress of the Verification .....	<b>3-30</b>
Completing the Verification .....	<b>3-31</b>
Stopping the Verification Before It Completes .....	<b>3-32</b>

## Reviewing Verification Results

# 4

<b>About Reviewing Verification Results Tutorial</b> .....	<b>4-2</b>
Overview .....	<b>4-2</b>
Before You Start .....	<b>4-2</b>
<b>Opening Verification Results</b> .....	<b>4-3</b>
Opening Run-Time Checks Perspective .....	<b>4-3</b>
Opening Verification Results .....	<b>4-3</b>
<b>Exploring the Viewer Window</b> .....	<b>4-4</b>
Overview .....	<b>4-4</b>
Reviewing Run-Time Checks Pane .....	<b>4-7</b>
<b>Reviewing Results in Manual Mode</b> .....	<b>4-9</b>
What Is Manual Mode? .....	<b>4-9</b>
Switching to Manual Mode .....	<b>4-9</b>
Reviewing Checks in Manual Mode .....	<b>4-9</b>
Reviewing Additional Examples of Checks .....	<b>4-15</b>
Filtering Checks .....	<b>4-19</b>
<b>Reviewing Results in Assistant Mode</b> .....	<b>4-24</b>
What Is Assistant Mode? .....	<b>4-24</b>
Switching to Assistant Mode .....	<b>4-24</b>
Selecting the Methodology and Criterion Level .....	<b>4-25</b>
Exploring Methodology for C++ .....	<b>4-25</b>
Reviewing Checks .....	<b>4-27</b>
Defining a Custom Methodology .....	<b>4-29</b>
<b>Generating Reports of Verification Results</b> .....	<b>4-31</b>

Polyspace Report Generator Overview .....	4-31
Generating Verification Reports .....	4-32

## Checking Compliance with Coding Rules

# 5

<b>About Checking Compliance with Coding Rules</b>	
<b>Tutorial</b> .....	5-2
Overview .....	5-2
Before You Start .....	5-3
<b>Setting Up Coding Rules Checking</b> .....	5-4
Opening Your Project .....	5-4
Creating New Verification .....	5-5
Setting the JSF++ Checking Option .....	5-7
Creating a JSF++ Rules File .....	5-7
Excluding Files from JSF++ Checking .....	5-10
Configuring Text and XML Editors .....	5-11
Saving the Project .....	5-12
<b>Running a Verification with Coding Rules Checking</b> ..	5-13
Starting the Verification .....	5-13
Examining JSF C++ Violations .....	5-15
Opening JSF Report .....	5-17

## 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 1-12
- “Working with Polyspace Software” on page 1-13
- “Additional Information and Support” on page 1-16
- “Related Products” on page 1-17

## Product Overview

In this section...
“Overview of Polyspace Verification” on page 1-2
“The Value of Polyspace Verification” on page 1-2

### Overview of Polyspace Verification

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

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

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

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

### The Value of Polyspace Verification

Polyspace verification can help you to:

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

## Ensure Software Reliability

Polyspace software ensures the reliability of your C++ applications by proving code correctness and identifying run-time errors. Using advanced verification techniques, Polyspace software performs an exhaustive verification of your source code.

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

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

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

You can also improve the quality of your code by using Polyspace software to check that your code complies with established C++ coding standards, such as MISRA® C++:2008 or JSF++:2005.

## Decrease Development Time

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

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

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

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

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

## **Improve the Development Process**

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

Polyspace verification software supports code verification throughout the development process:

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

## Product Components

In this section...
“Polyspace Products for C++” on page 1-5
“Polyspace Verification Environment” on page 1-5
“Other Polyspace Components” on page 1-10

### Polyspace Products for C++

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

- Polyspace® Client™ for C/C++
- Polyspace® Server™ for C/C++

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

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

### Polyspace Verification Environment

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

The Polyspace verification environment consists of three perspectives:

- “Project Manager Perspective” on page 1-6
- “Coding Rules Perspective” on page 1-8
- “Run-Time Checks Perspective” on page 1-9

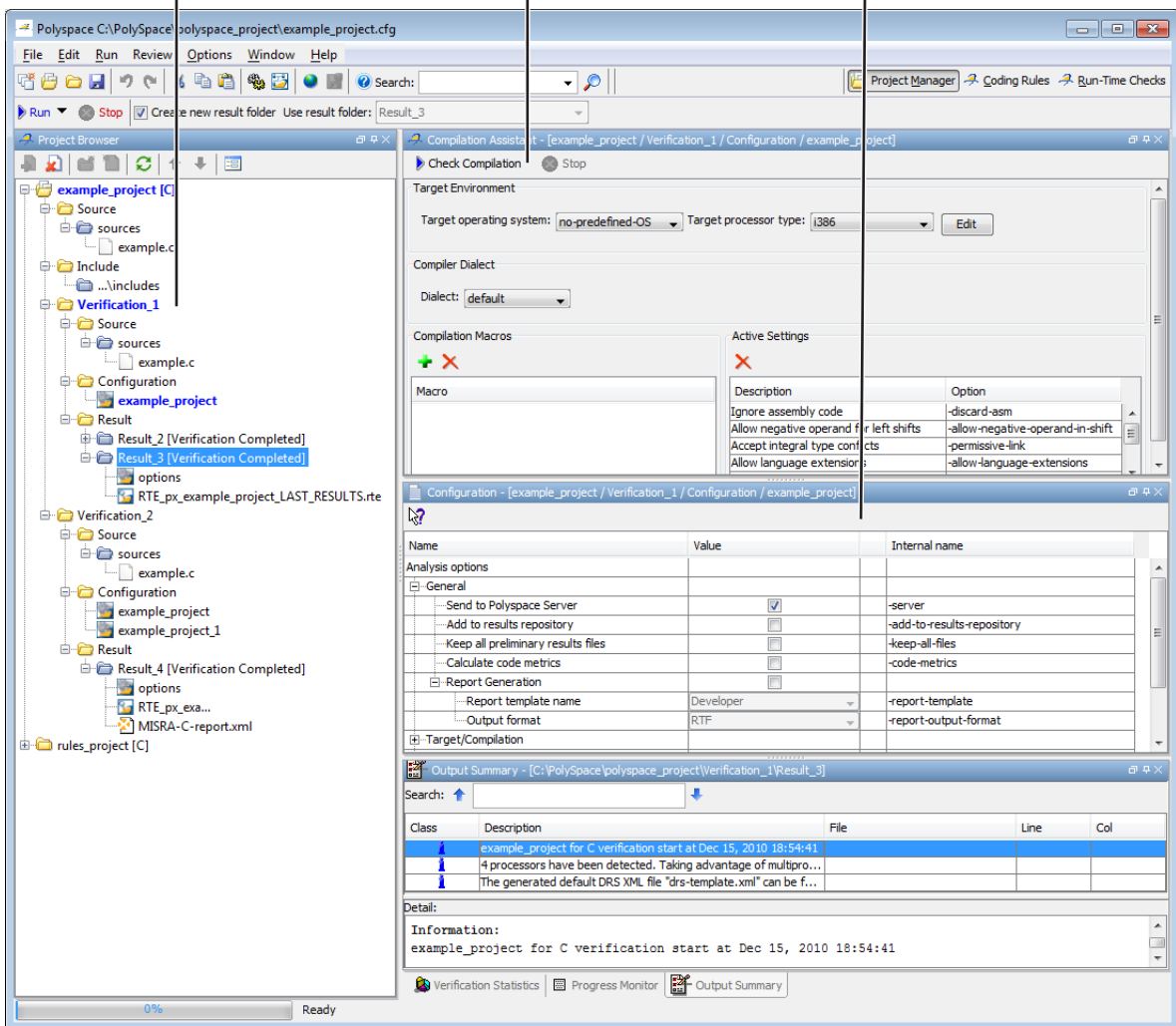
## **Project Manager Perspective**

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

Specify source files and include folders

Set target environment and check compilation

Specify analysis options

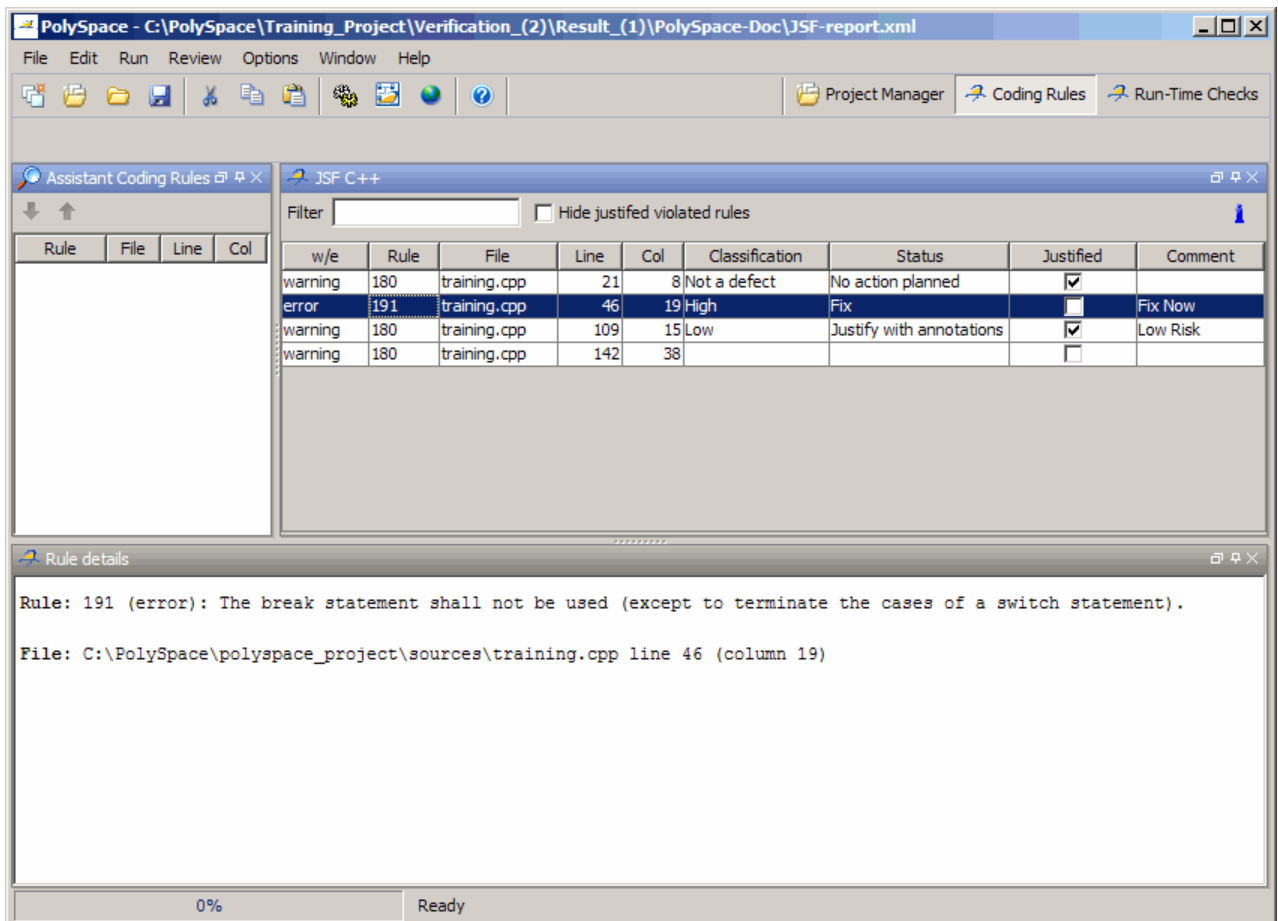


Monitor progress and view logs

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

## Coding Rules Perspective

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



You use the Coding Rules perspective in the tutorial in Chapter 5, “Checking Compliance with Coding Rules”.



## Run-Time Checks Perspective

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

Review Details
Review Statistics

The screenshot displays the PolySpace IDE interface with the Run-Time Checks perspective. The main window shows the source code of `training.cpp` with a red error message: "the training.MathUtils::Recursion(int\*) call never terminates". The Run-Time Checks pane on the left lists various checks, with "NNT.6" highlighted in red. The Review Statistics pane shows a table of coding review progress.

Coding review progress	Count	Progress
Red NTC justified / to justify	0/1	0
Red justified / to justify	0/2	0
Gray justified / to justify	0/1	0
Orange justified / to justify	1/6	16
Software reliability indicator	79/88	89

Calls	Line
training.MathUtils::Recursion_caller()	117
pst_stubs.Utils::random_int()	120
pst_stubs.Utils::random_int()	123
training.MathUtils::Recursion(int*)	124
pst_stubs.Utils::random_int()	127
training.MathUtils::Recursion(int*)	128
_polyspace_main.main	110

Variables	# R#	# Write	W.T.	R.T.	Line
polyspace_main.polyspace_0	0	1			1_pol
polyspace_main_init					1_pol

Run-Time Checks
Source code
Variable Access
Call Hierarchy

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

## Other Polyspace Components

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

- Polyspace Queue Manager Interface (Spooler)
- Polyspace in One Click
- Polyspace Metrics Web Interface

### Polyspace Queue Manager Interface (Polyspace Spooler)

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

The screenshot shows a window titled "PolySpace Queue Manager Interface" with a menu bar containing "Operations" and "Help". Below the menu bar is a table with the following columns: ID, Author, Application, Results folder, CPU, Status, Date, and Language. The table contains four rows of data. The last row is highlighted in blue. At the bottom of the window, it says "Connected to Queue Manager localhost" on the left and "User mode" on the right.

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	Training_Project	C:\PolySpace\polyspace_project\results	runstr...	running	06-Jan-2010, ...	CPP

Connected to Queue Manager localhost User mode

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

### Polyspace in One Click

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

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

You use Polyspace in One Click in the tutorial “Using Polyspace In One Click to Launch Verification” on page 3-21.

### **Polyspace Metrics Web Interface**

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

For information on using Polyspace Metrics, see “Software Quality with Polyspace Metrics” in the *Polyspace Products for C++ User’s Guide*.

## Installing Polyspace Products

In this section...
“Finding the Installation Instructions” on page 1-12
“Obtaining Licenses for Polyspace® Client for C/C++ and Polyspace® Server for C/C++ Products” on page 1-12

### Finding the Installation Instructions

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

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

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

# Working with Polyspace Software

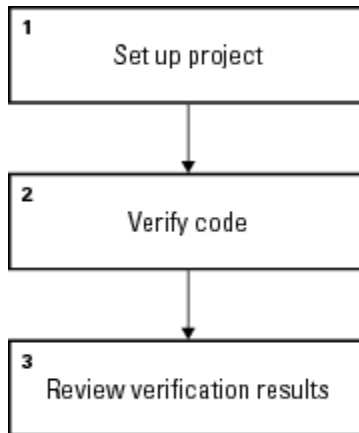
## In this section...

“Basic Workflow” on page 1-13

“Tutorials in This Guide” on page 1-14

## Basic Workflow

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



In this workflow, you:

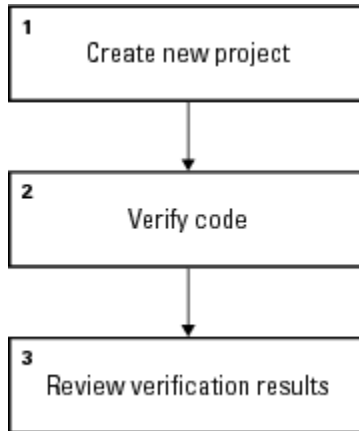
- 1** Use the Project Manager perspective to set up a project file.
- 2** Verify code on a server or client.

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

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

## Tutorials 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 will follow in this guide is:



In this workflow, you:

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

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

- 2** Verify a single class using demo C++ source code.

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

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

- 3** Review the verification results.

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

- 4** Modify the project to include coding rules checking and review the JSF++ rule violations in the example file.

This step is in the tutorial in Chapter 5, “Checking Compliance with Coding Rules”.

## Additional Information and Support

In this section...
“Product Help” on page 1-16
“MathWorks Online” on page 1-16

### Product Help

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

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

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

### MathWorks Online

For additional information and support, see:

[www.mathworks.com/products/polyspace](http://www.mathworks.com/products/polyspace)



## Related Products

In this section...
“Polyspace Products for Verifying C Code” on page 1-17
“Polyspace Products for Verifying Ada Code” on page 1-17
“Polyspace Products for Linking to Models” on page 1-17

### **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 Polyspace Project

---

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

# About Setting Up a Project Tutorial

In this section...
“Overview” on page 2-2
“Example Files” on page 2-2

## Overview

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

## Example Files

In this tutorial, you will verify the class `MathUtils` in the source file `training.cpp` that comes with the Polyspace installation CD. You can 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 Polyspace Verification Environment” on page 2-5

“Creating a New Project to Verify a Class in the Training C++ File” on page 2-7

### What Is a Project?

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

- Source files
- Include folders
- One or more configurations, specifying a set of analysis options
- One or more verifications, each of which include:
  - Source (specific versions of source files used in the verification)
  - Configuration (specific set of analysis options used for the verification)
  - Verification results

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

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

### Preparing the Project Folders

Before you start verifying C++ code 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 create separate folders for the source files and include files within the project folder.

For this tutorial, prepare a project folder as follows:

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

- 3 Copy the file `training.cpp` from

```
Polyspace_Install\Examples\Demo_Cpp_Long\sources
```

to

```
polyspace_project\sources
```

where *Polyspace\_Install* is the installation folder.

- 4 Copy the files `training.h` and `zz_utils.h` from

```
Polyspace_Install\Examples\Demo_Cpp_Long\sources
```

to

```
polyspace_project\includes.
```

## Opening Polyspace Verification Environment

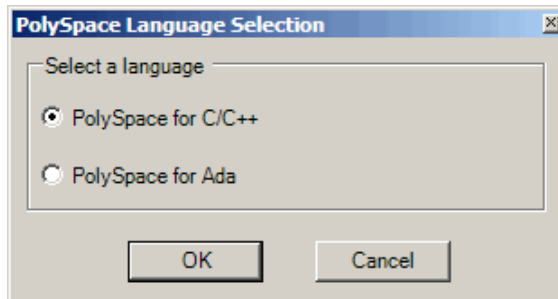
You use the Polyspace verification environment to create projects, start verifications, and review verification results.

To open the Polyspace verification environment:

- 1 Double-click the **Polyspace** icon.



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



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

The Polyspace Verification Environment opens.

## 2 Setting Up a Polyspace® Project

Specify source files  
and include folders

Set target environment  
and check compilation

Specify  
analysis options

The screenshot displays the Polyspace IDE interface with three main panels. The left panel shows a Project Browser with a tree view of the project structure, including source files and folders. The middle panel shows the Configuration Assistant dialog, where the Target Environment is set to 'no-predefined-OS' and the Target processor type is 'i386'. The right panel shows the Configuration dialog, where Analysis options are configured, including 'Send to Polyspace Server' and 'Report Generation'. The bottom panel shows the Output Summary window, which displays a table of messages and a detail view of the first message.

Name	Value	Internal name
Analysis options		
General		
Send to Polyspace Server	<input checked="" type="checkbox"/>	-server
Add to results repository	<input type="checkbox"/>	-add-to-results-repository
Keep all preliminary results files	<input type="checkbox"/>	-keep-all-files
Calculate code metrics	<input type="checkbox"/>	-code-metrics
Report Generation		
Report template name	Developer	-report-template
Output format	RTF	-report-output-format
Target/Compilation		

Class	Description	File	Line	Col
	example_project for C verification start at Dec 15, 2010 18:54:41			
	4 processors have been detected. Taking advantage of multipro...			
	The generated default DRS XML file "drs-template.xml" can be f...			

Detail:  
Information:  
example\_project for C verification start at Dec 15, 2010 18:54:41

Monitor progress and view logs



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

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

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

## Creating a New Project to Verify a Class in the Training 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 to verify `training.cpp`.

You create a new project by:

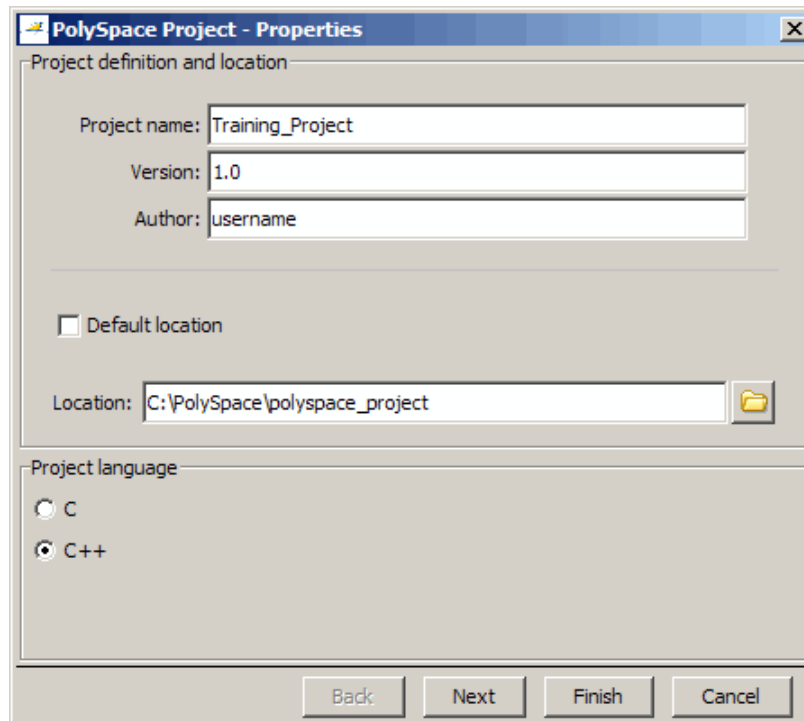
- “Opening a New project” on page 2-7
- “Specifying Source Files and Include Folders” on page 2-10
- “Specifying Target Environment” on page 2-11
- “Specifying Analysis Options” on page 2-12
- “Saving the Project” on page 2-15

### Opening a New project

To open a new project for verifying `training.cpp`:

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

The Polyspace Project – Properties dialog box opens:



### 2 In the **Project name** field, enter Training\_Project.

### 3 Clear the **Default location** check box.

---

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

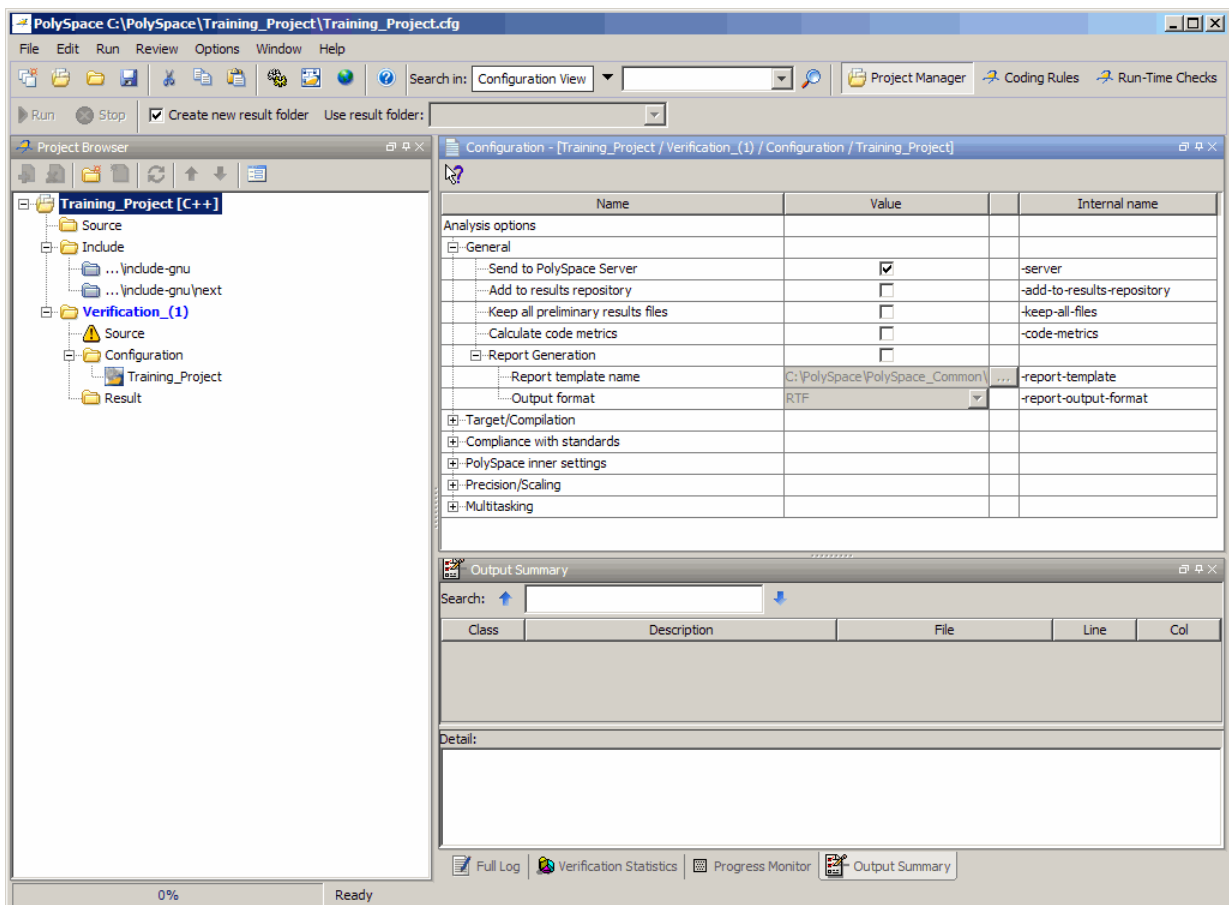
---

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

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

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


The `Training_Project` opens in the Polyspace verification environment.



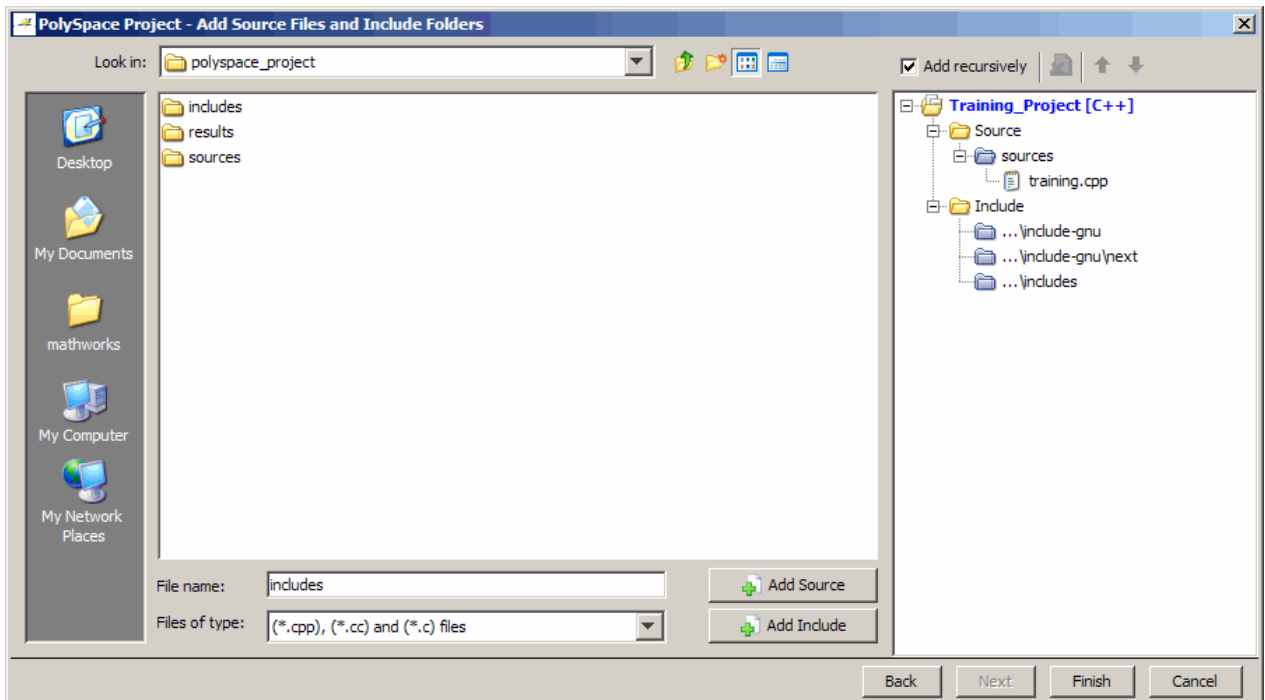
### Specifying Source Files and Include Folders

To specify the source files and include folders for the verification of `training.cpp`:

1 In the Project Browser, select the Source folder.

2 Click the **Add source** icon  in the upper left of the Project Browser.

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



3 The project folder `polyspace_project` should appear in **Look in**. If it does not, navigate to that folder.

4 Select the `sources` folder, then click **Add Source**.

The `training.cpp` file appears in the Source tree for `Training_Project`.

- 5 Select the `includes` folder, then click **Add Include**.

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

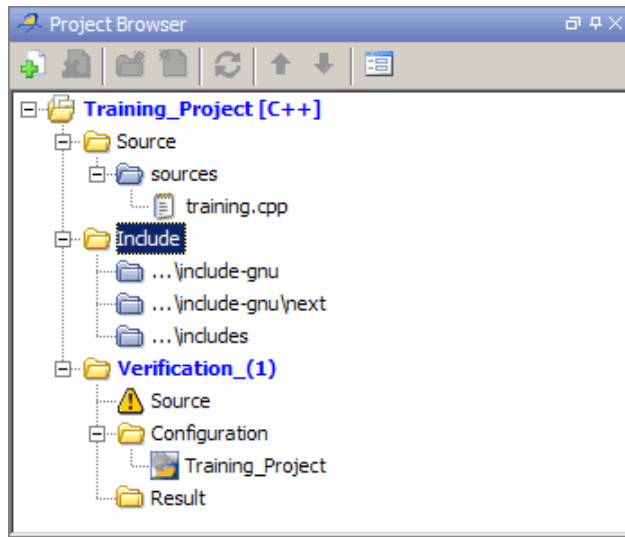
---

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

---

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

The Project Browser now looks like the following graphic.



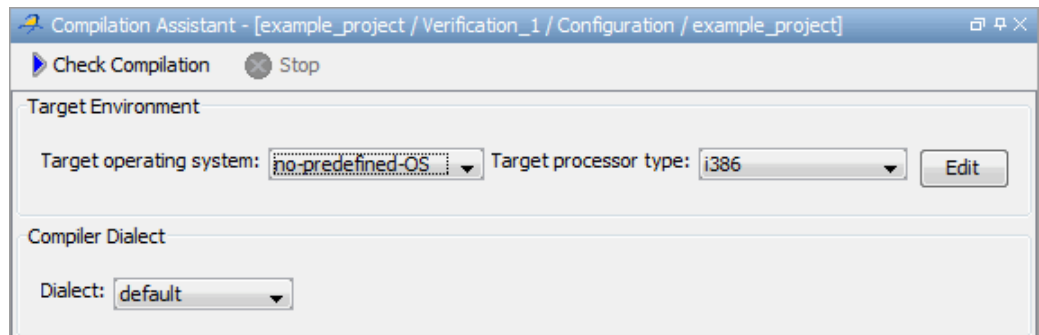
## Specifying Target Environment

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

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

To specify the target environment for this tutorial:

- 1 In the **Target operating system** drop-down menu, select `no_predefined_OS`.



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

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

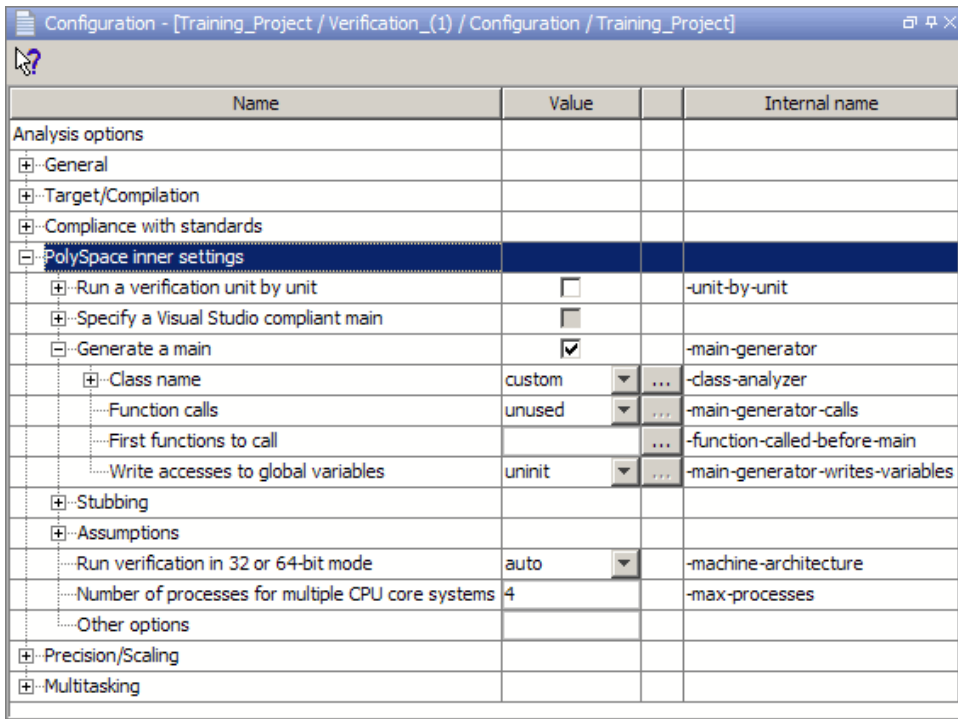
### Specifying Analysis Options

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

For 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 Configuration pane, expand the **Polyspace inner settings** section.



**2** Select the **Generate a main** check box.

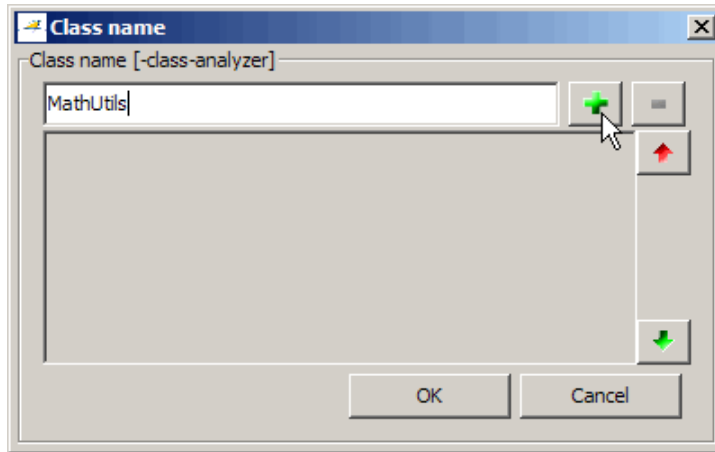
This enables the `-class-analyzer` option and allows you to specify the class you want to verify.


**3** Expand the **Generate a main** section.

**4** In the **Class name** drop-down menu, select `custom`.

**5** Select the browse button in the Class name row.

**6** The Class name dialog box opens.



**7** Enter `MathUtils` in the Class name field, then select the **Add item**  button.

**8** `MathUtils` is added to the list of classes.

**9** Click **OK**. to save your changes and close the dialog box.

**10** Keep the default values for all other analysis options.

---

**Note** You can also select the `-class-only` option when you want to verify a single class. When this option is applied, even if you add other classes and function member definitions, Polyspace will stub them. This accelerates your verification process and allows you to check robustness issues for a single class. For the purposes of this tutorial, it is not necessary to select this option because the class `MathUtils` does not depend on any other classes.

---



## **Saving the Project**

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

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



# Running a Verification

---

- “About Running a Verification Tutorial” on page 3-2
- “Preparing for Verification” on page 3-4
- “Launching Server Verification from Project Manager” on page 3-10
- “Using Polyspace In One Click to Launch Verification” on page 3-21
- “Launching Client Verification from Project Manager” on page 3-28

## About Running a Verification Tutorial

In this section...
“Overview” on page 3-2
“Before You Start” on page 3-3

### Overview

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

<b>Use...</b>	<b>For...</b>
Project Manager	A basic way to start a verification. You specify the source files in the project file. With the project 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 how to start a verification using the Project Manager and Polyspace In One Click. You verify the class `MathUtils` in the file `training.cpp` three times using a different method each time. You use:

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

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

## Before You Start

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

## Preparing for Verification

In this section...
“Opening the Project” on page 3-4
“Specifying Source Files to Verify” on page 3-4
“Checking for Compilation Problems” on page 3-5

### Opening the Project

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

To open `Training_Project.cfg`:

**1** If the Polyspace software is not already open, open it.

**2** Select **File > Open Project**.

The Open a Polyspace project file dialog box opens.

**3** In the **Look in** drop-down list box, navigate to `polyspace_project`.

**4** Select `Training_Project.cfg`.

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

### Specifying Source Files to Verify

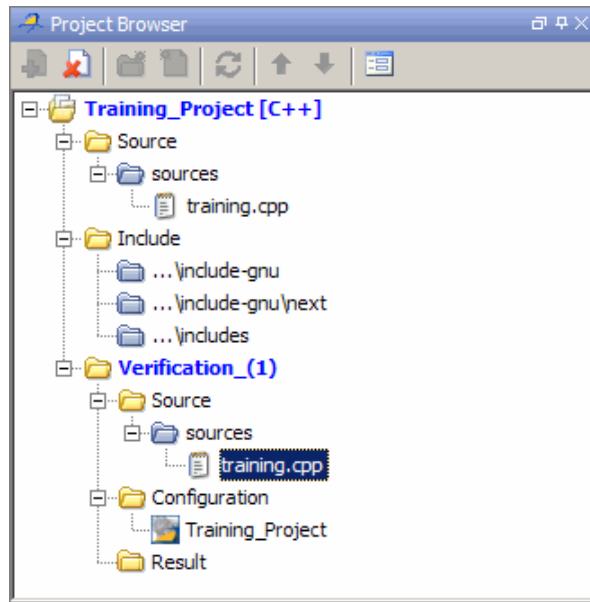
Each Polyspace project can contain multiple verifications. Each of these verifications can analyze a specific set of source files using a specific set of analysis options.

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

To copy source files to a verification:

- 1 In the Project Browser Source tree, right click `training.cpp`.
- 2 Select **Copy Source File to > Verification\_(1)**.

The `training.cpp` file appears in the Source tree of `Verification_(1)`.



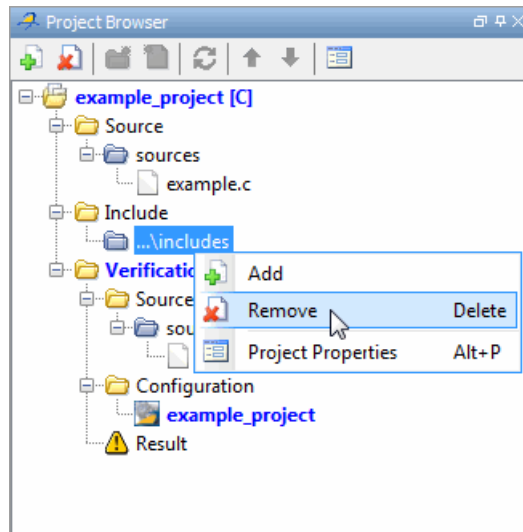
## Checking for Compilation Problems

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

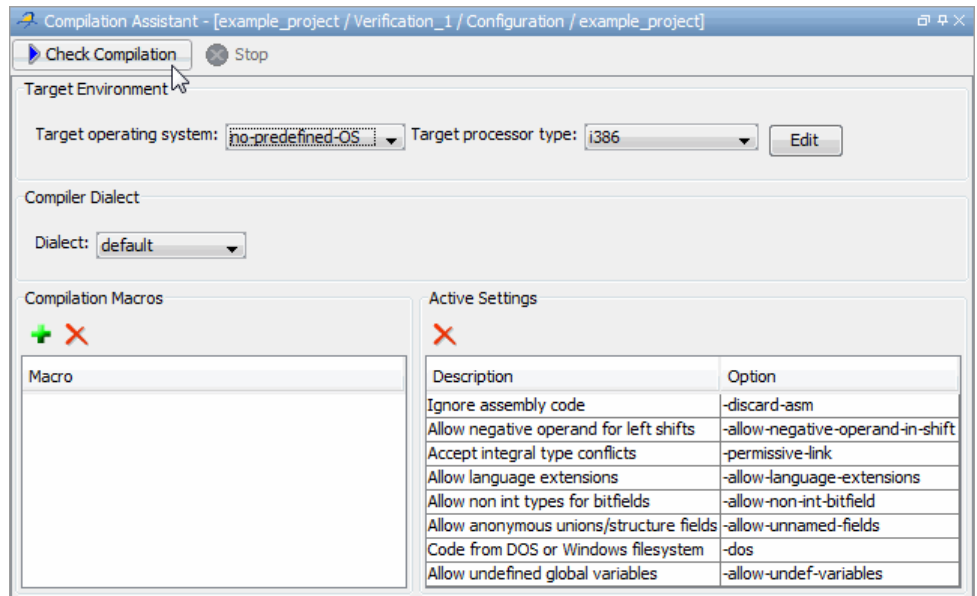
To check your project for compilation problems:

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

### 3 Running a Verification

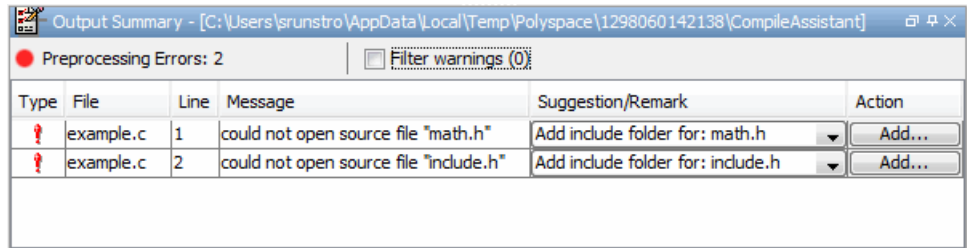


2 In the Compilation Assistant window, click **Check Compilation**.



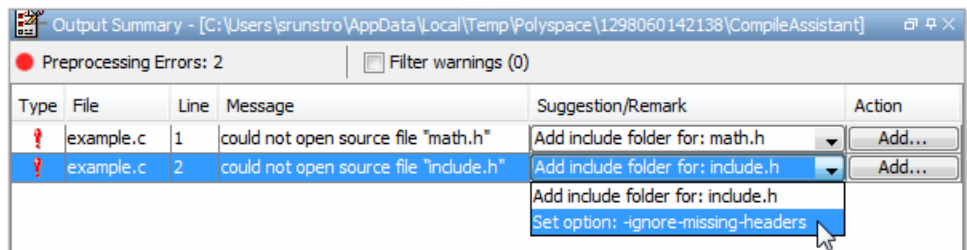


The software compiles your code and checks for errors, and reports the results in the Output Summary.



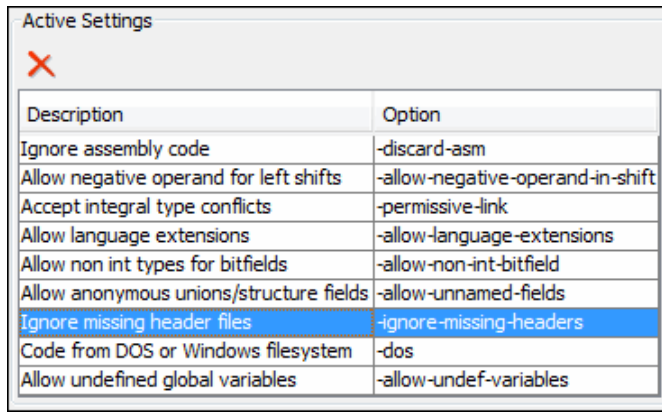
Because you removed the include folder, the software reports a compilation error for the project, along with suggested solutions for the problem.

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



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

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



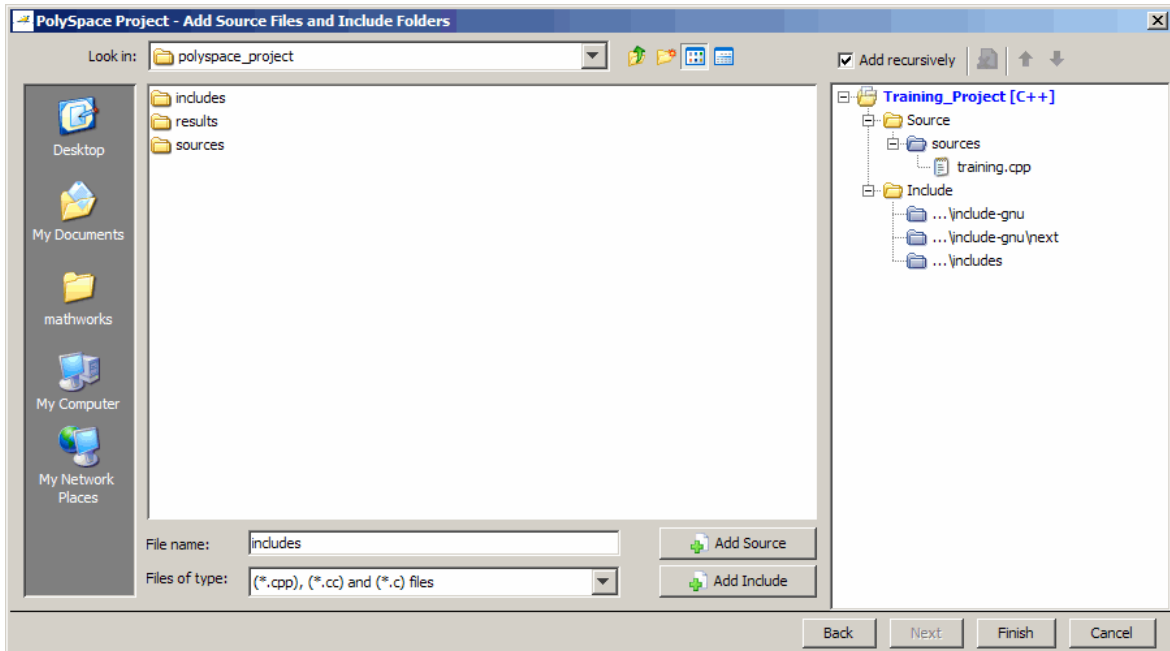
Description	Option
Ignore assembly code	-discard-asm
Allow negative operand for left shifts	-allow-negative-operand-in-shift
Accept integral type conflicts	-permissive-link
Allow language extensions	-allow-language-extensions
Allow non int types for bitfields	-allow-non-int-bitfield
Allow anonymous unions/structure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

**6** Select **Check Compilation** to check your project again.

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

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

The Add Source Files and Include Folders dialog box opens.



**8** If necessary, navigate to the polyspace\_project folder.

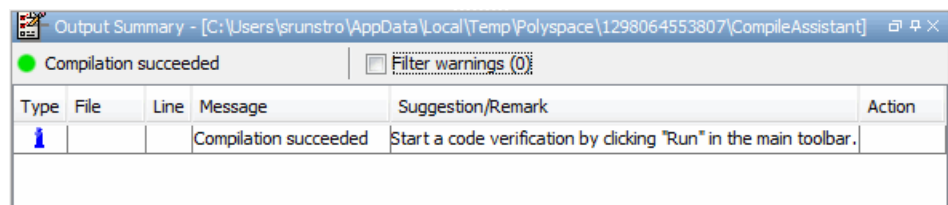
**9** Select the includes folder, then click **Add Include**.

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

**10** Click **Finish**.

**11** Select **Check Compilation** in the Compilation Assistant to check your project again.

The message `Compilation succeeded` appears in the Output Summary.



## Launching Server Verification from Project Manager

In this section...
“Starting the Verification” on page 3-10
“Monitoring Progress of the Verification” on page 3-12
“Removing Verification Results from the Server” on page 3-18
“Troubleshooting a Failed Verification” on page 3-19


### Starting the Verification

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

To start a verification that runs on a server:

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

Name	Value	Internal name
Analysis options		
[-] General		
Send to PolySpace Server	<input checked="" type="checkbox"/>	-server
Add to results repository	<input type="checkbox"/>	-add-to-results-repository
Keep all preliminary results files	<input type="checkbox"/>	-keep-all-files
Calculate code metrics	<input type="checkbox"/>	-code-metrics
[-] Report Generation		
Report template name	C:\PolySpace\...	-report-template
Output format	RTF	-report-output-format

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

---

**Note** If you see the message Verification process failed, click **OK** and go to “Troubleshooting a Failed Verification” on page 3-19.

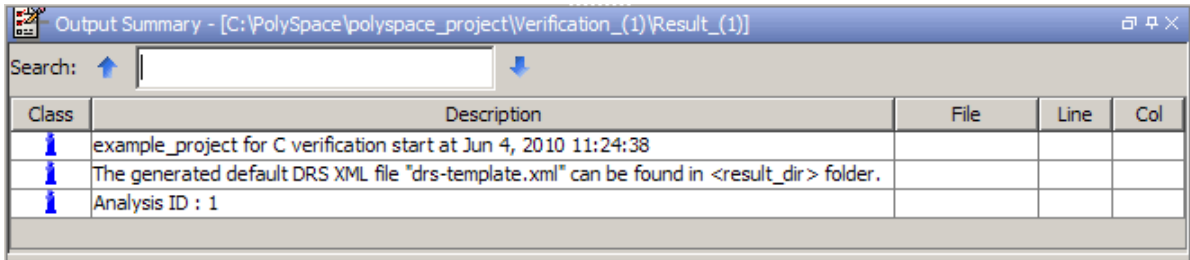
---

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 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 is complete:

- You see the message `queued on server` at the bottom of the Project Manager perspective. This message indicates that the part of the verification that takes place on the client is complete. The rest of the verification runs on the server.
- A message in the Output Summary window gives you the identification number (Analysis ID) for the verification. For this verification, the identification number is 1.



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

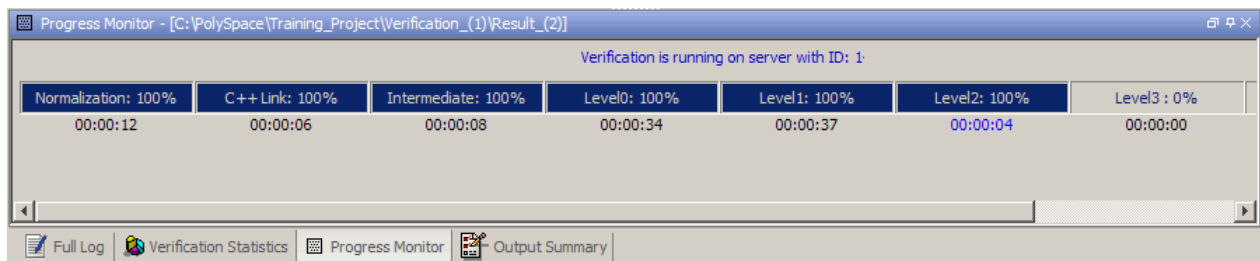
## Monitoring Progress of the Verification

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

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

## Monitoring Progress Using Project Manager

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



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

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

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

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

---

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

---

### Monitoring Progress Using Queue Manager

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

To monitor the verification of Training\_Project:

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




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	Training_Project	C:\PolySpace\polyspace_project\results	runstr...	running	06-Jan-2010, ...	CPP

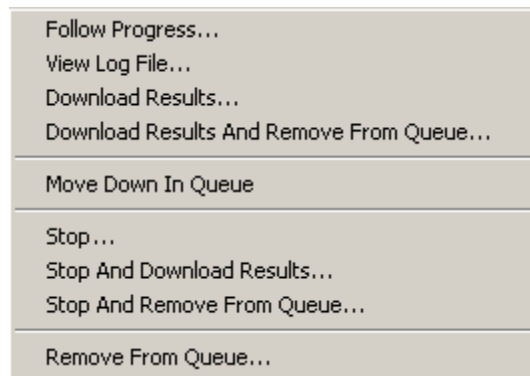
Connected to Queue Manager localhost User mode

---

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

---

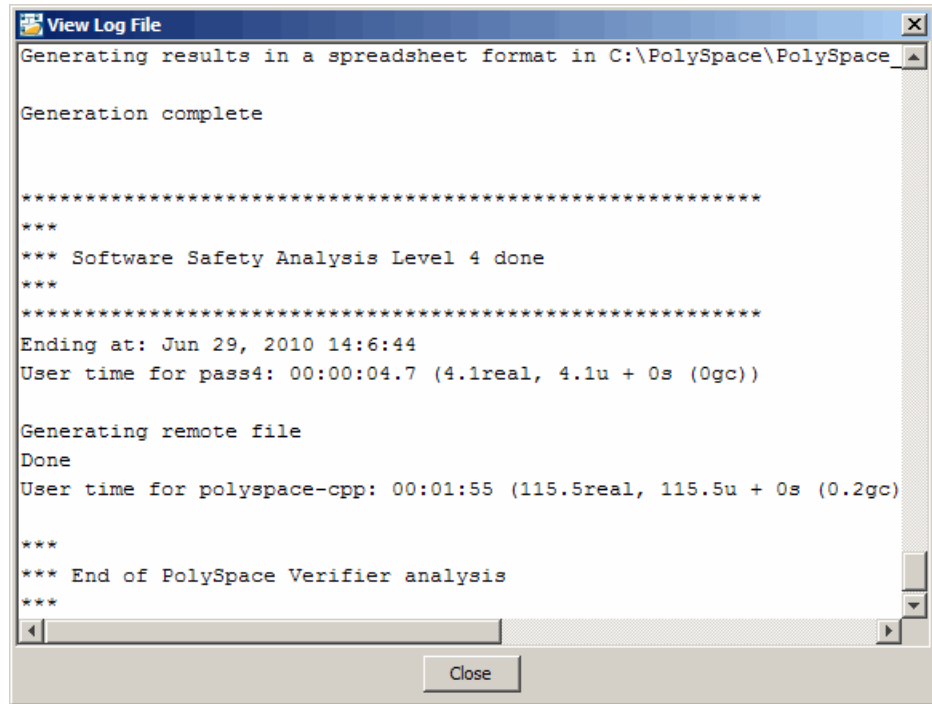
- 2 Point anywhere in the row for your job (ID 15 in the figure above).
- 3 Right-click to open the context menu for this verification.



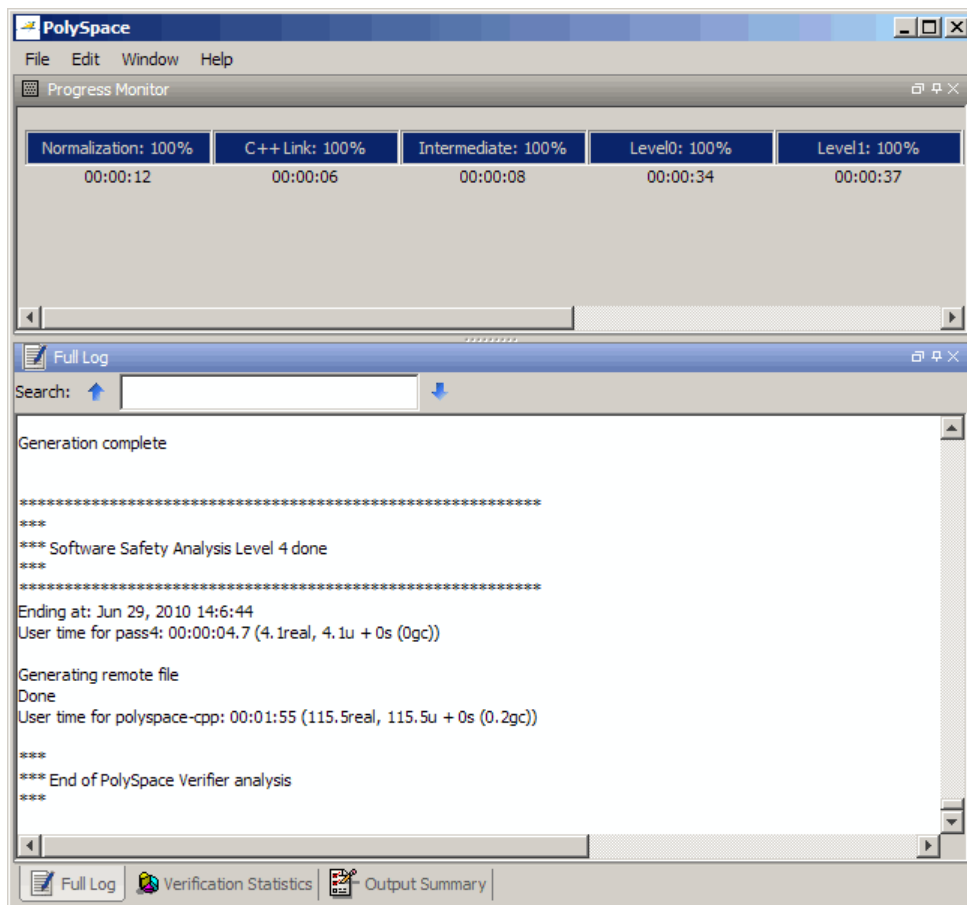
- 4 Select **View log file**.

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






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



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

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

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

---

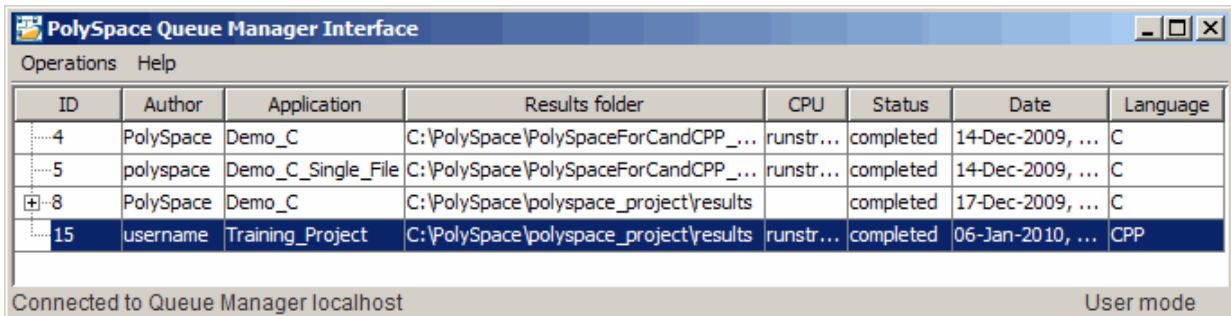
**Note** You can search the logs. In the **Search in the log** box, enter a search term and click the left arrows to search backward or the right arrows to search forward.

---

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

**8** Wait for the verification to finish.

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



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	Training_Project	C:\PolySpace\polyspace_project\results	runstr...	completed	06-Jan-2010, ...	CPP

Connected to Queue Manager localhost User mode

## Removing Verification Results from the Server

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

---

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

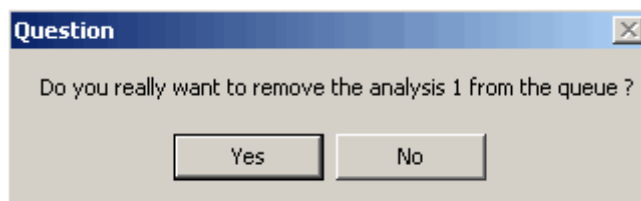
---

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

To remove your results from the server:

- 1 In the Polyspace Queue Manager Interface, right-click the verification, and select **Remove From Queue**.

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



- 2 Click **Yes**.

---

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

---

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

Once the results are on your client, you can review them using the Run-Time Checks perspective. You review 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](http://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 a Class in the Training C++ File” on page 2-7.

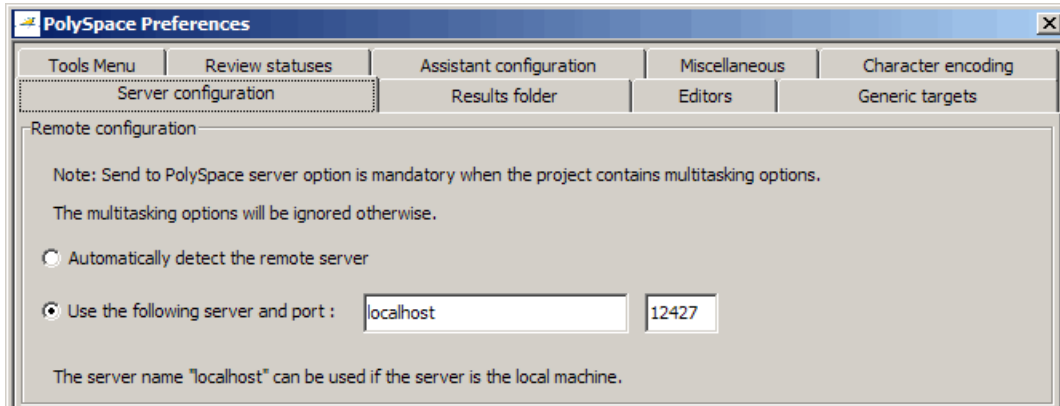
## Polyspace Software Cannot Find the Server

If you see the following message in the log, Polyspace software cannot find the server.

Error: Unknown host :

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

- 1 Select **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 Launch Verification

### In this section...

“Overview of Polyspace In One Click” on page 3-21

“Setting the Active Project” on page 3-21

“Sending the Files to Polyspace Software” on page 3-23

### Overview of Polyspace In One Click

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

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

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

### Setting the Active Project

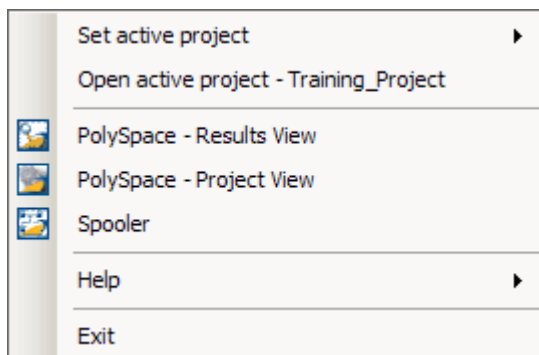
The active project is the project that Polyspace In One Click uses to verify the files that you select. Once you have set an active project, it remains active until you change the active project. Polyspace software uses the analysis options from the project; it does not use the source files or results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

To set the active project:

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

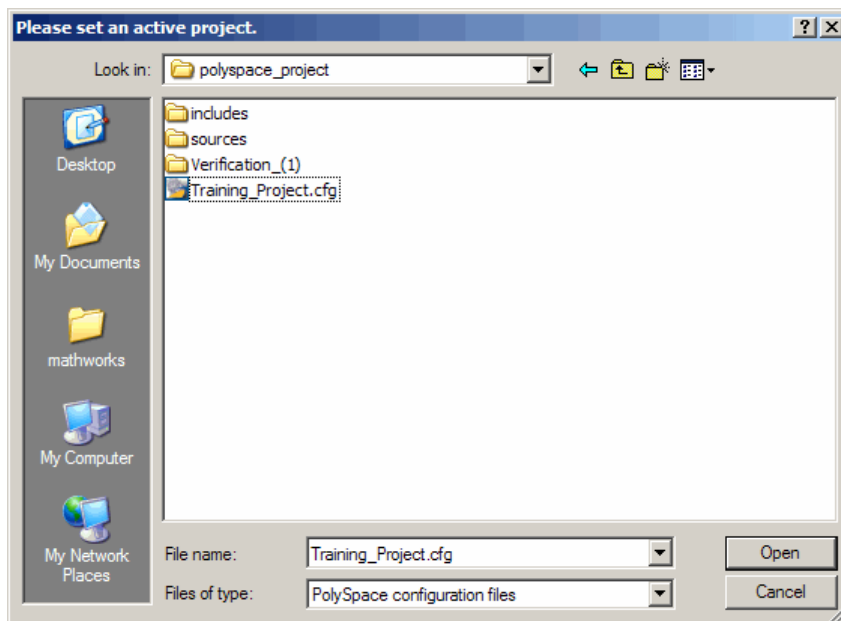


The context menu appears.



**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 `Training_Project.cfg`.
- 5 Click **Open** to apply the changes and close the dialog box.

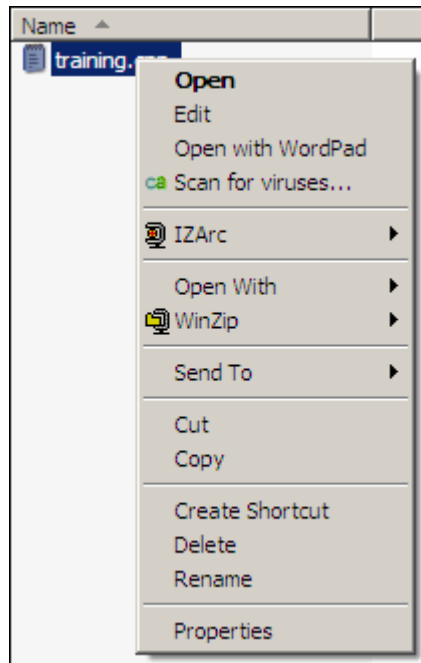
## Sending the Files to Polyspace Software

You can send several files to Polyspace software for verification. For this tutorial, you send one file, `training.cpp`.

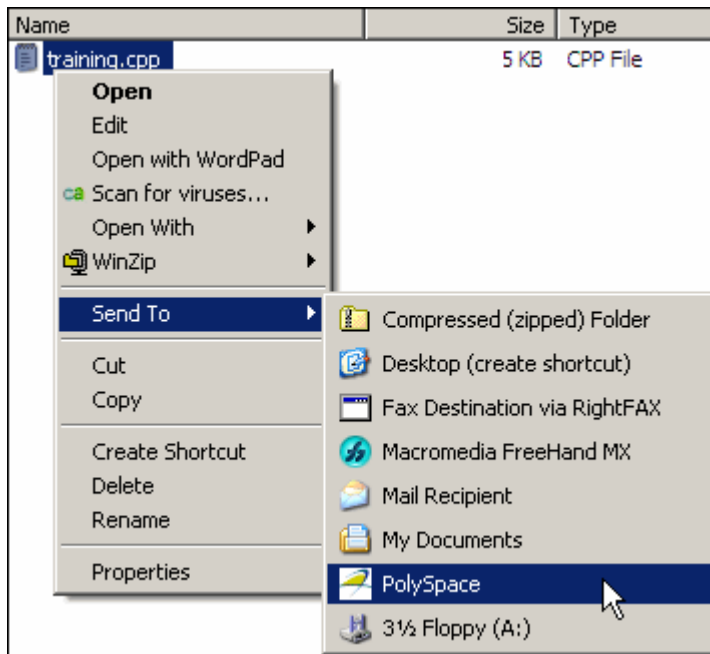
To send `training.cpp` to Polyspace software for verification:

- 1 Navigate to the folder `polyspace_project\sources`.
- 2 Right-click the file `training.cpp`.

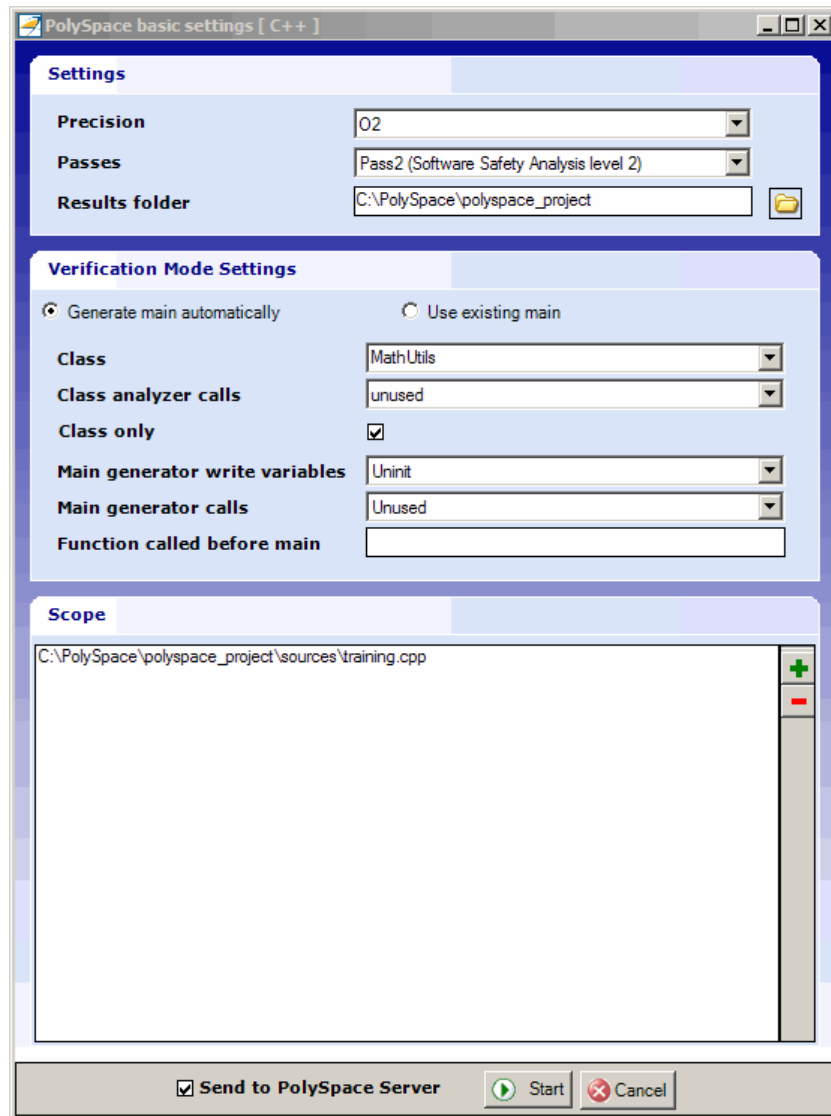
The context menu appears.



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



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



- 4 Make sure that **Results folder** is `polyspace_project\results`.
- 5 The Verification Mode Settings are inherited from the active project (`Training_Project.cfg`). For this tutorial, you are verifying a single class

(MathUtils), which you already configured in the project. Therefore, you do not need to modify these parameters.

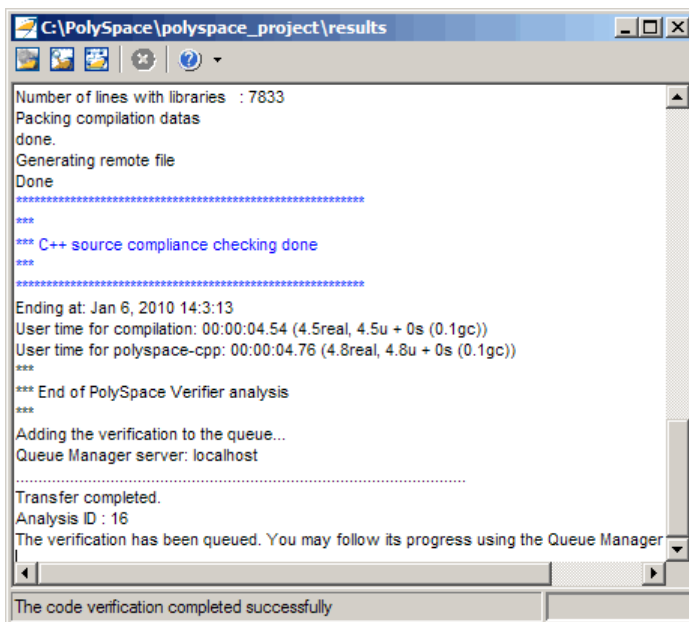
---

**Note** The **Class only** checkbox is selected by default. This activates the `-class-only` option in Polyspace. For the purposes of this tutorial, it does not matter whether or not this option is applied because the class `MathUtils` does not depend on any other classes.

---

- 6 Select the **Send to Polyspace Server** option if it is not already selected.
- 7 Leave the default values for the other parameters.
- 8 Click **Start**.

The verification log appears.



```
C:\PolySpace\polyspace_project\results
Number of lines with libraries : 7833
Packing compilation datas
done.
Generating remote file
Done
*****
***
*** C++ source compliance checking done
***
*****
Ending at: Jan 6, 2010 14:3:13
User time for compilation: 00:00:04.54 (4.5real, 4.5u + 0s (0.1gc))
User time for polyspace-cpp: 00:00:04.76 (4.8real, 4.8u + 0s (0.1gc))
***
*** End of PolySpace Verifier analysis
***
Adding the verification to the queue...
Queue Manager server: localhost
.....
Transfer completed.
Analysis ID : 16
The verification has been queued. You may follow its progress using the Queue Manager
The code verification completed successfully
```

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

- You see the following message in the log:

End of Polyspace Verifier analysis

- A message in the log tells you that the verification was transferred to the server queue and gives you the identification number (Analysis ID) for the verification. For this verification, the identification number is 1.
- Monitor the verification using the Spooler. For information on using the Spooler to monitor a verification on a server, see “Monitoring Progress Using Queue Manager” on page 3-13.
- When the verification completes, download the results to `polyspace_project\results`. For information on downloading results from a server to a client, see “Removing Verification Results from the Server” on page 3-18

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

# Launching Client Verification from Project Manager

In this section...
“Starting the Verification” on page 3-28
“Monitoring the Progress of the Verification” on page 3-30
“Completing the Verification” on page 3-31
“Stopping the Verification Before It Completes” on page 3-32

## Starting the Verification

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

---

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

---

To start a verification that runs on a client:


- 1 If the project `Training_Project.cfg` is not already open, open the project.

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

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

Name	Value	Internal name
Analysis options		
[-] General		
Send to PolySpace Server	<input checked="" type="checkbox"/>	-server
Add to results repository	<input type="checkbox"/>	-add-to-results-repository
Keep all preliminary results files	<input type="checkbox"/>	-keep-all-files
Calculate code metrics	<input type="checkbox"/>	-code-metrics
[-] Report Generation	<input type="checkbox"/>	
Report template name	C:\PolySpace\... ..	-report-template
Output format	RTF	-report-output-format

**3** If you see a warning that multitasking is not available when you run a verification on the client, click **OK** to continue and close the message box. This warning only appears when you clear the **Send to Polyspace Server** check box.

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

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

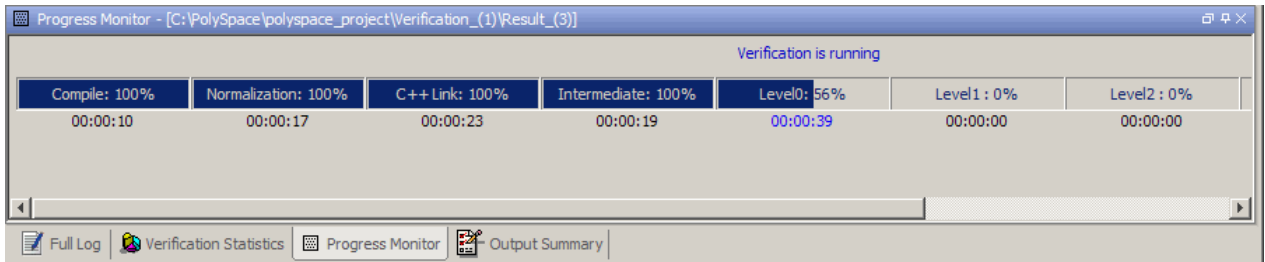
---

**Note** If you see the message *Verification process failed*, click **OK** and go to “Troubleshooting a Failed Verification” on page 3-19.

---


## 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 Project Manager window. Follow the next steps to view the logs:

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

---

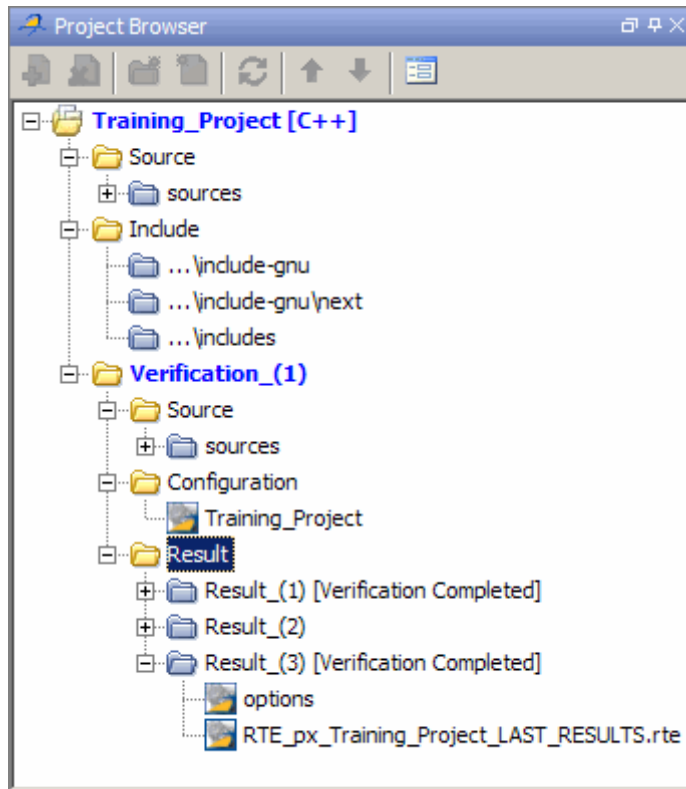
**Note** You can search the logs. In the **Search in the log** box, enter a search term and click the arrows to search backward or forward.

---



## Completing the Verification

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



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

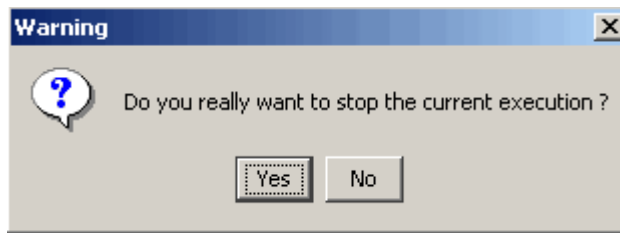
## Stopping the Verification Before It Completes

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

To stop a verification:

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

A warning dialog box opens.



- 2 Click **Yes**.

The verification stops and the message Verification process stopped appears.

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

---

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

---

# Reviewing Verification Results

---

- “About Reviewing Verification Results Tutorial” on page 4-2
- “Opening Verification Results” on page 4-3
- “Exploring the Viewer Window” on page 4-4
- “Reviewing Results in Manual Mode” on page 4-9
- “Reviewing Results in Assistant Mode” on page 4-24
- “Generating Reports of Verification Results” on page 4-31

## About Reviewing Verification Results Tutorial

In this section...
“Overview” on page 4-2
“Before You Start” on page 4-2

### Overview

In the previous tutorial, Chapter 3, “Running a Verification” , you completed a verification of the class `MathUtils` in the file `training.cpp`. In this tutorial, you explore the verification results.

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

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

### Before You Start

Before starting this tutorial, complete the tutorial Chapter 3, “Running a Verification”. In this tutorial, you use the verification results stored in this file:

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

## Opening Verification Results

### In this section...

“Opening Run-Time Checks Perspective” on page 4-3

“Opening Verification Results” on page 4-3

### Opening Run-Time Checks Perspective

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

To open the Run-Time Checks perspective:

- Select the **Run Time Checks** button  in the Polyspace Verification Environment toolbar.

### Opening Verification Results

To open the verification results:

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

The Please select a file dialog box opens.

- 2** Navigate to the results folder:

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

- 3** Select the file RTE\_px\_Training\_Project\_LAST\_RESULTS.rte.

- 4** Click **Open**.

The results appear in the Run-Time Checks perspective.

---

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

---

## Exploring the Viewer Window

In this section...
“Overview” on page 4-4
“Reviewing Run-Time Checks Pane” on page 4-7

### **Overview**

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

Review Details Review Statistics

The screenshot displays the PolySpace Viewer interface with the following components:

- Run-Time Checks:** A tree view on the left showing various checks like 'NNT.0' through 'NNT.9' and 'EXC.0' through 'EXC.9'. A red 'NNT.6' is highlighted.
- Review Details:** A central panel showing details for a specific issue at line 124, column 13. It includes classification (High), status (Fix), and a comment: 'the training.MathUtils::Recursion(int\*) call never terminates'. It also notes an unreachable check.
- Source code:** A code editor at the bottom showing the implementation of `MathUtils::Recursion_caller()` with a recursive call to `Recursion(x)`.
- Review Statistics:** A table in the top right showing progress for various NTC types.
- Call Hierarchy:** A table in the middle right showing the call stack for `training.MathUtils::Recursion_caller()`.
- Variable Access:** A table in the bottom right showing variable access for `polyspace_main_polyspace_0`.

Run-Time Checks Source code Variable Access Call Hierarchy

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

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

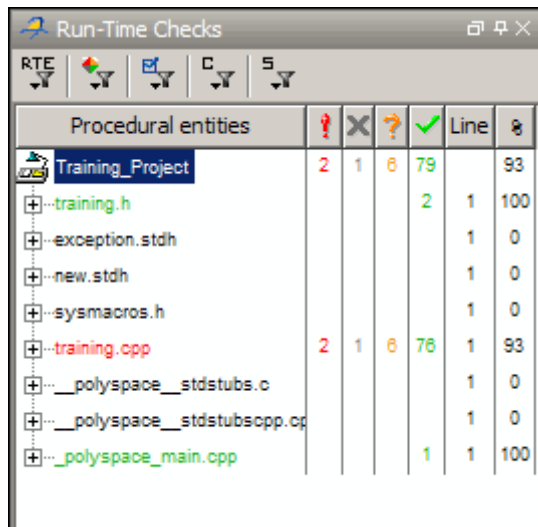
You can resize or hide any of these sections. You learn more about the Run-Time Checks perspective later in this tutorial.



## Reviewing Run-Time Checks Pane






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

When you first open the results file from the verification of `training.cpp`, the Run-Time Checks pane looks like this.



Procedural entities	RTE	Warning	Error	Warning	Error	Line	%
Training_Project	2	1	8	79		93	
+-training.h				2	1	100	
+-exception.stdh					1	0	
+-new.stdh					1	0	
+-sysmacros.h					1	0	
+-training.cpp	2	1	8	76		93	
+-__polyspace__stdstubs.c					1	0	
+-__polyspace__stdstubs.cpp					1	0	
+-__polyspace__main.cpp				1	1	100	

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

Column 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.

---

**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 Manual Mode

### In this section...

- “What Is Manual Mode?” on page 4-9
- “Switching to Manual Mode” on page 4-9
- “Reviewing Checks in Manual Mode” on page 4-9
- “Reviewing Additional Examples of Checks” on page 4-15
- “Filtering Checks ” on page 4-19

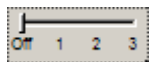
### What Is Manual Mode?

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

### Switching to Manual Mode

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

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



The toolbar displays buttons and menus specific to expert mode.

### Reviewing Checks in Manual Mode

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

- “Selecting a Check to Review” on page 4-10
- “Displaying the Calling Sequence” on page 4-11
- “Tracking Review Progress” on page 4-12

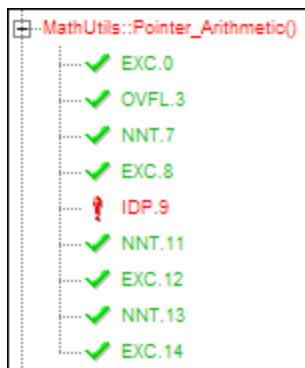
### Selecting a Check to Review

In the procedural entities view, `training.cpp` is red, indicating that this file has at least one red check.

To review a red check in `training.cpp`:

- 1 In the procedural entities section of the Run-Time Checks pane, expand `training.cpp`.
- 2 Expand the red procedure `MathUtils::Pointer_Arithmetic()`.

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

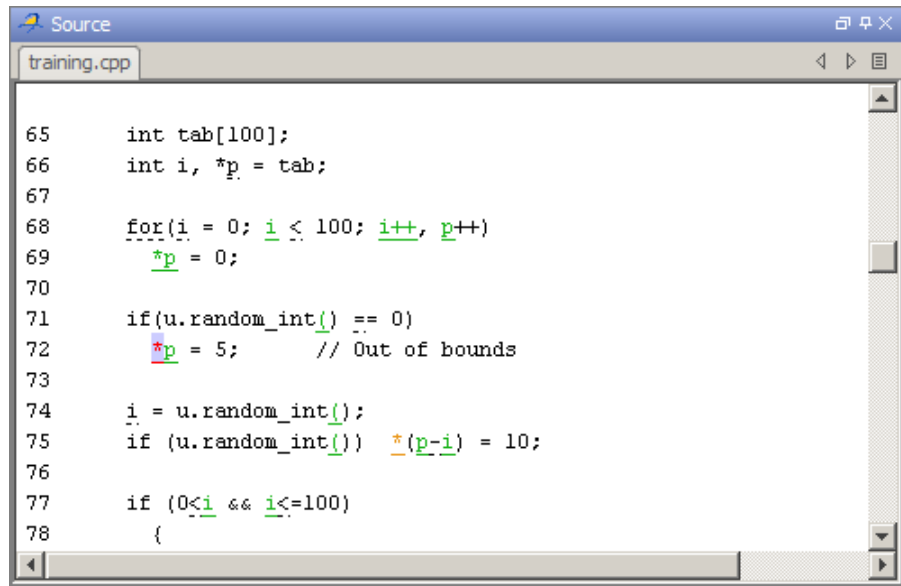


Each item in the list of checks has an acronym that identifies the type of check and a number. For example, in `IDP.9`, `IDP` stands for `Illegal Dereferenced Pointer`.

For more information about different types of checks, see “Check Descriptions” in the *Polyspace Products for C++ Reference*.

- 3 Click on the red `IDP.9`.

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



```

65     int tab[100];
66     int i, *p = tab;
67
68     for(i = 0; i <= 100; i++, p++)
69         *p = 0;
70
71     if(u.random_int() == 0)
72         *p = 5;      // Out of bounds
73
74     i = u.random_int();
75     if (u.random_int()) *(p-i) = 10;
76
77     if (0<i && i<=100)
78         {

```

- 4 At line 72 of the code, click on the red code.


An error message box appears indicating that when the pointer `p` is dereferenced, it is outside of its bounds. At line 66, `p` points to the start of `tab` which has 100 elements. The for loop starting at line 68 initializes the elements of `tab` to 0. This for loop leaves `p` pointing to the location after the last element of `tab`.

### Displaying the Calling Sequence

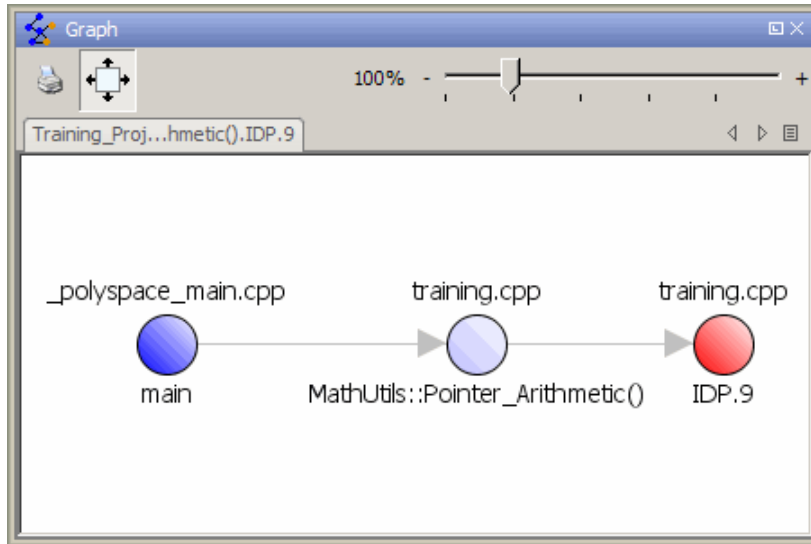
You can display the calling sequence that leads to the code associated with a check. To see the calling sequence for the red IDP.9 check in `MathUtils::Pointer_Arithmetic()`:

- 1 Expand `MathUtils::Pointer_Arithmetic()`.

- 2 Click the red IDP.9.

- 3 Click the **call graph** button  in the Review Details toolbar.

A window displays the call graph.



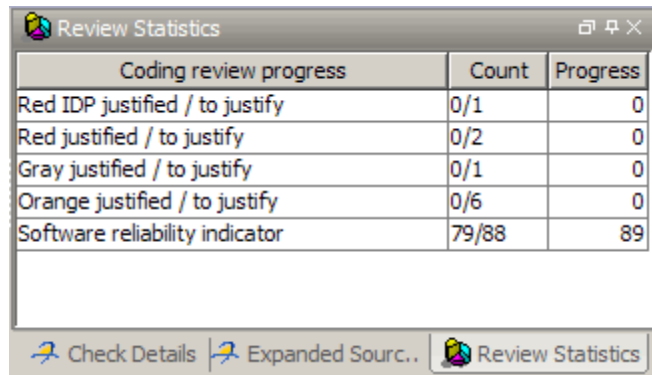
The code associated with IDP.9 is in `MathUtils::Pointer_Arithmetic`. The generated main function calls `MathUtils::Pointer_Arithmetic`.

### 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.9 check in `MathUtils::Pointer_Arithmetic()`:

- 1 Expand `MathUtils::Pointer_Arithmetic()`.
- 2 Click the red IDP.9.

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



Coding review progress	Count	Progress
Red IDP justified / to justify	0/1	0
Red justified / to justify	0/2	0
Gray justified / to justify	0/1	0
Orange justified / to justify	0/6	0
Software reliability indicator	79/88	89

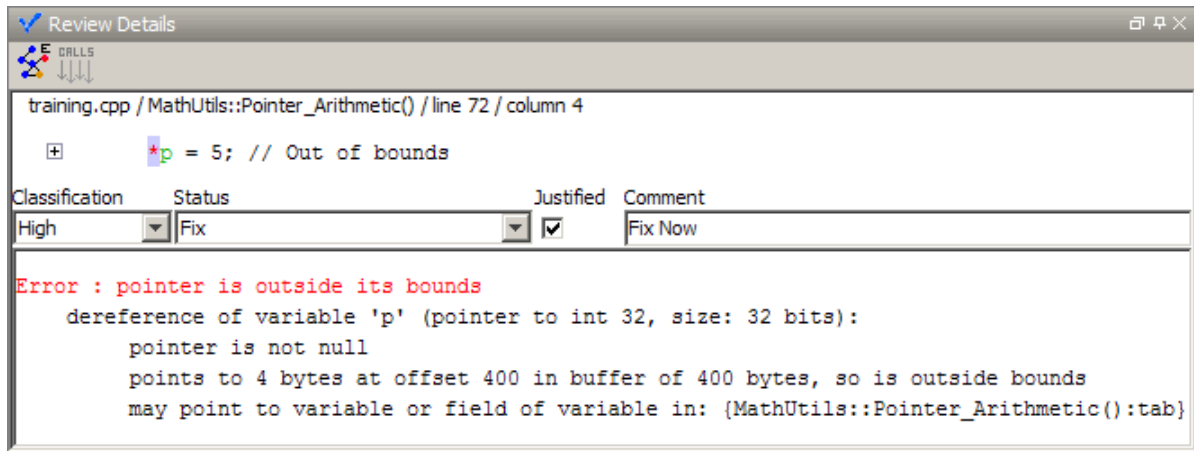
The **Count** column displays a ratio and the **Progress** column displays the equivalent percentage.

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

The last row displays the ratio of the number of green checks to the total number of checks, providing an indicator of the reliability of the software.

Information about the current check (the red IDP.9) appears in the upper-right part of the Viewer window.



**3** After you review the check, select a **Classification** to describe the seriousness of the issue:

- High
- Medium
- Low
- Not a defect

**4** Select a **Status** to describe how you intend to address the issue:

- Fix
- Improve
- Investigate
- Justify with annotations
- No Action Planned
- Other
- Restart with different options
- Undecided



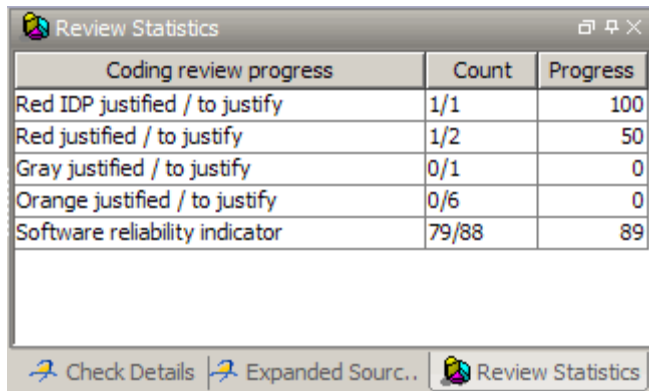
---

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

---

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

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



Coding review progress	Count	Progress
Red IDP justified / to justify	1/1	100
Red justified / to justify	1/2	50
Gray justified / to justify	0/1	0
Orange justified / to justify	0/6	0
Software reliability indicator	79/88	89

Check Details | Expanded Sourc.. | Review Statistics

## 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 `training.cpp`:

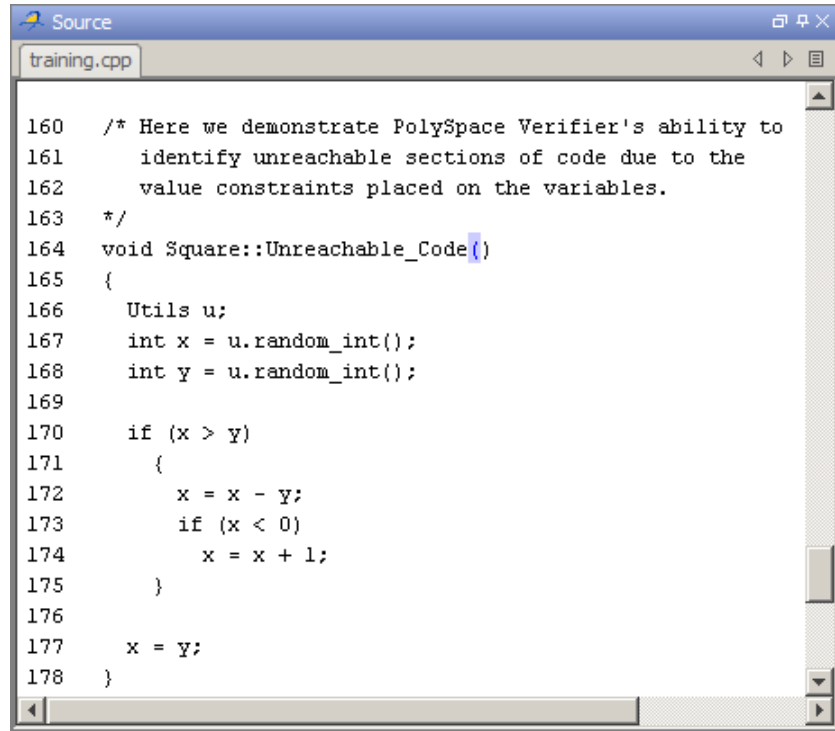
- “Example: Unreachable Code” on page 4-15
- “Example: A Function with No Errors” on page 4-17
- “Example: Division by Zero” on page 4-18

### Example: Unreachable Code

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

**1** In **Procedural Entities**, click on `Square::Unreachable_Code()`.

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



```
160  /* Here we demonstrate PolySpace Verifier's ability to
161     identify unreachable sections of code due to the
162     value constraints placed on the variables.
163  */
164  void Square::Unreachable_Code()
165  {
166     Utils u;
167     int x = u.random_int();
168     int y = u.random_int();
169
170     if (x > y)
171     {
172         x = x - y;
173         if (x < 0)
174             x = x + 1;
175     }
176
177     x = y;
178 }
```

**2** Examine the source code.

At line 174, the code `x = x + 1` is never reached because the condition `x < 0` is always false.

---

**Note** In the **Procedural Entities** view all public and protected member functions for the classes `RTE` and `Square` are marked as unreachable code. This is because the analysis results are from the single class verification of `MathUtils` which does not depend on any other classes.

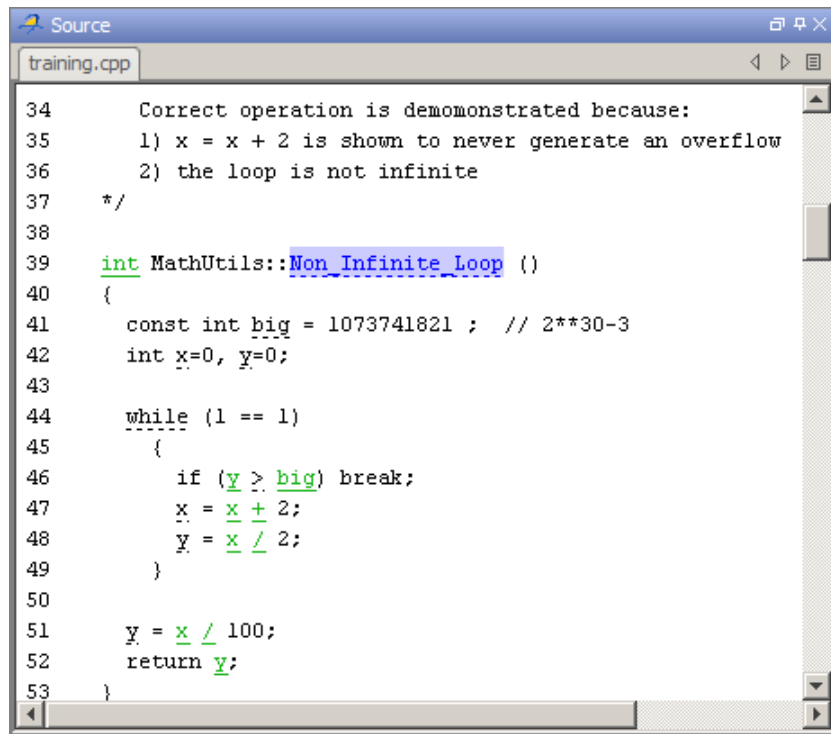
---

### 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 on the green `MathUtils::Non_Infinite_Loop()` function.

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



```
34     Correct operation is demomonstrated because:
35     1) x = x + 2 is shown to never generate an overflow
36     2) the loop is not infinite
37     */
38
39     int MathUtils::Non_Infinite_Loop ()
40     {
41         const int big = 1073741821 ; // 2**30-3
42         int x=0, y=0;
43
44         while (1 == 1)
45         {
46             if (y > big) break;
47             x = x + 2;
48             y = x / 2;
49         }
50
51         y = x / 100;
52         return y;
53     }
```

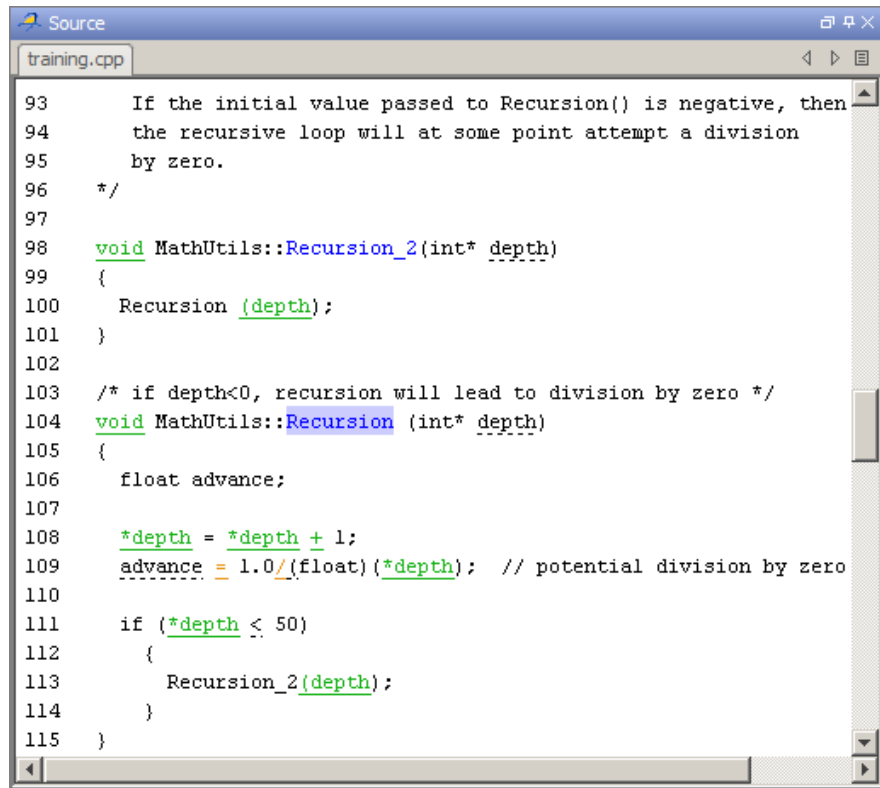
- 2 Examine the source code. The variable `x` never overflows because the `while` loop at line 44 terminates before `x` can overflow.

### Example: Division by Zero

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

- 1 In **Procedural entities**, expand `MathUtils::Recursion()`.

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



```
93     If the initial value passed to Recursion() is negative, then
94     the recursive loop will at some point attempt a division
95     by zero.
96     */
97
98     void MathUtils::Recursion_2(int* depth)
99     {
100         Recursion (depth);
101     }
102
103     /* if depth<0, recursion will lead to division by zero */
104     void MathUtils::Recursion (int* depth)
105     {
106         float advance;
107
108         *depth = *depth + 1;
109         advance = 1.0/(float)(*depth); // potential division by zero
110
111         if (*depth <= 50)
112         {
113             Recursion_2(depth);
114         }
115     }
```

- 2 Examine the `MathUtils::Recursion()` function.

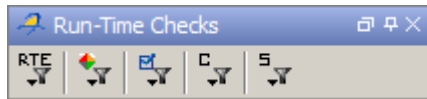
When `Recursion()` is called with `depth` less than zero, the code at line 109 will result in division by zero. The orange color indicates that this is a potential error (depending on the value of `depth`).

## Filtering Checks

You can filter the checks that you see in the Run-Time Checks perspective so that you can focus on certain checks. Polyspace software allows you to filter your results in several ways. You can filter by:

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

To filter checks, select one of the filter buttons in the Run-Time checks toolbar.




---

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

---

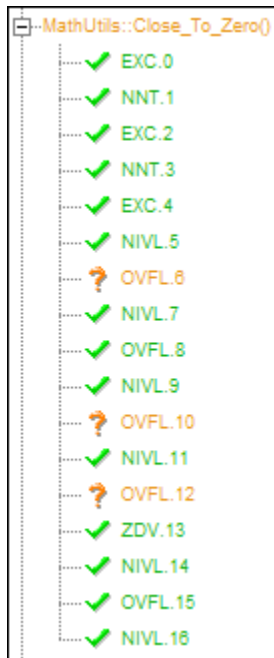
### Example: Filtering NIVL Checks


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

To filter NIVL checks:

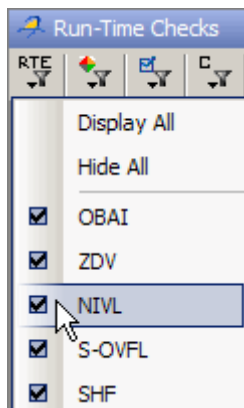
- 1 Expand `MathUtils::Close_To_Zero()`.

`Square_Root()` has 16 checks: 13 are green, and three are orange.

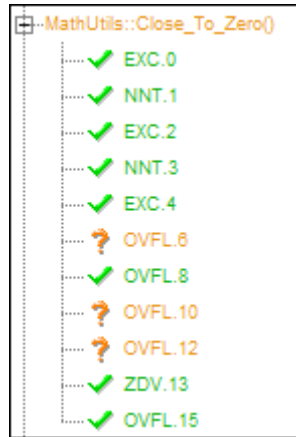


2 Click the **RTE filter** icon .

3 Clear the **NIVL** option.



The software hides the NIVL checks for `MathUtils::Close_To_Zero()`.



**4** Select the **NIVL** option to redisplay the NIVL checks.

---

**Note** When you filter a check category, some red checks within that category are still displayed. For example, if you filter IDP checks, IDP.9 is still displayed under `MathUtils::Pointer_Arithmetic()`.

---

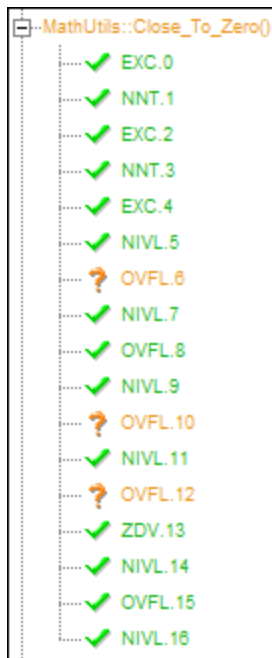
### Example: Filtering Green Checks

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

To filter green checks:

- 1 Expand `MathUtils::Close_To_Zero()`.

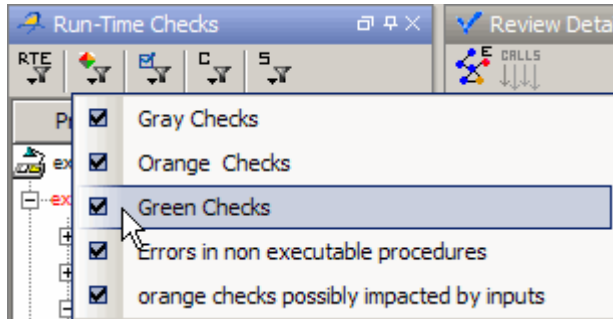
`Square_Root()` has 16 checks: 13 are green, and three are orange.



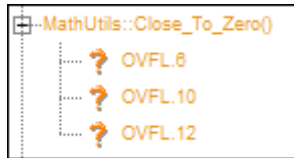
- 2 Click the **Color filter** icon .

- 3 Clear the **Green Checks** option.





The software hides the green checks.



## Reviewing Results in Assistant Mode

### In this section...

- “What Is Assistant Mode?” on page 4-24
- “Switching to Assistant Mode” on page 4-24
- “Selecting the Methodology and Criterion Level” on page 4-25
- “Exploring Methodology for C++” on page 4-25
- “Reviewing Checks” on page 4-27
- “Defining a Custom Methodology” on page 4-29

### What Is Assistant Mode?

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

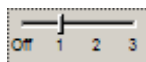
- 1 All red checks.
- 2 All blocks of gray checks (the first check in each unreachable function).
- 3 Orange checks according to the methodology and criterion level you select.

You will learn about methodologies and criterion levels in “Selecting the Methodology and Criterion Level” on page 4-25.

### Switching to Assistant Mode

To switch from expert to assistant mode:

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



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



The controls for assistant mode include:

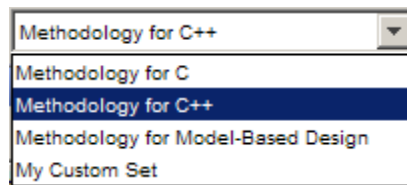
- A menu for selecting the review methodology for orange checks.
- A slider for selecting the criterion level within that methodology.
- Arrows for navigating through the reviews.

## Selecting the Methodology and Criterion Level

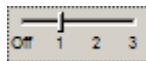
A methodology defines which orange checks you review in assistant mode. Each methodology has three criterion levels, corresponding to different development phases, with increasing review requirements. As the criterion level increases, you review more checks.

To select the methodology and level for this tutorial:

- 1 Select **Methodology for C++** from the methodology menu.



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



## Exploring Methodology for C++

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

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

- 1 Select **Options > Preferences**.

The Polyspace Preferences dialog box opens.

**2** Select the **Assistant configuration** tab.

The configuration for Methodology for C++ appears.

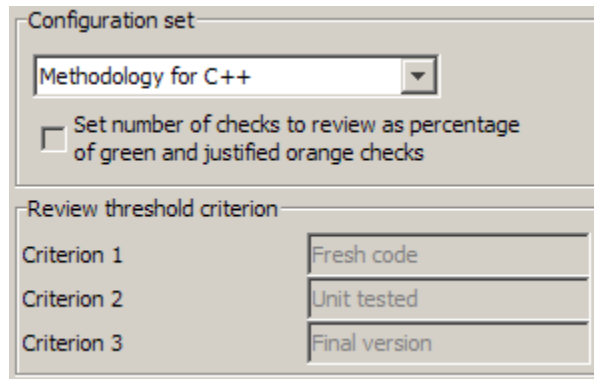
On the right side of the dialog box, a table shows the number of orange checks that you review for a given criterion and check category.

The screenshot shows the 'Assistant configuration' tab of the Polyspace Preferences dialog box. It features a table titled 'Number of checks to review' with columns for 'Criterion 1', 'Criterion 2', and 'Criterion 3'. The table is divided into two sections: 'Common' and 'C & C++ only'. The 'Common' section lists checks ZDV, NIVL, S-OVFL, COR, NIV, F-OVFL, and ASRT. The 'C & C++ only' section lists checks OBAI, SHF, IDP, and NIP. The table specifies the number of checks to review for each criterion and category.

	Criterion 1	Criterion 2	Criterion 3
<b>Common</b>			
ZDV	5	20	ALL
NIVL	10	50	ALL
S-OVFL	10	50	ALL
COR		10	10
NIV		5	10
F-OVFL	5	10	20
ASRT		5	20
<b>C &amp; C++ only</b>			
OBAI	10	20	ALL
SHF	5	10	ALL
IDP		10	20
NIP		10	20

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

In the lower-left part of the dialog box, the section **Review threshold criterion** contains text that appears in the tooltip for the criterion slider on the Run-Time Checks toolbar.



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

Criterion	Name in the Tooltip
1	Fresh code
2	Unit tested
3	Final version

These names correspond to phases of the development process.


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

## Reviewing Checks

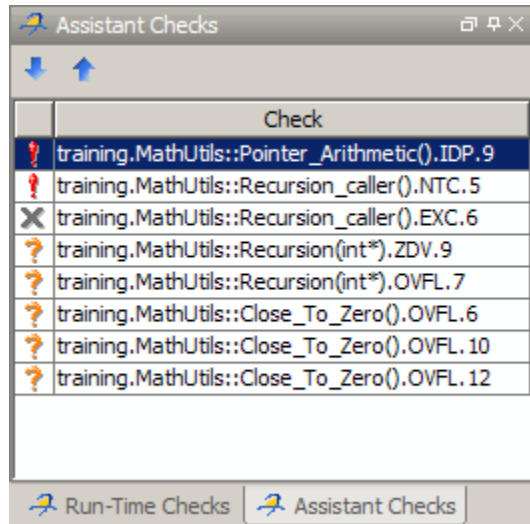
In assistant mode, you review checks in the order in which Polyspace software presents them:

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

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

- 1 Click the forward arrow .

The Assistant Checks tab shows the current check (IDP.9).



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

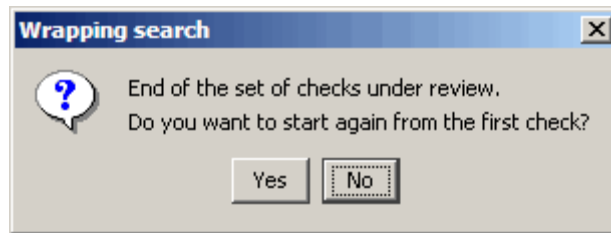
---

**Note** You can display the calling sequence and track review progress as described in “Reviewing Results in Manual Mode” on page 4-9.

---

- 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.

To define your custom methodology:

- 1 Select **Options > Preferences**.

The Polyspace Preferences dialog box opens.

- 2 Select the **Assistant configuration** tab.

- 3 Select **Add a set** from the **Configuration set** menu.


- 4 In the Create a new set dialog box, enter **My methodology** for the name and click **Enter** to close the dialog box.

- 5 Under the **Criterion 1** column, enter the number **1** next to **IDP**. Polyspace software selects up to one orange IDP for review.

The software will not select any other orange checks for review because you are leaving all of the other fields blank. This does not affect the red and gray checks. The software will still present all red checks and the first check in each unreachable function 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.3 check. You did not review IDP.3 earlier in the tutorial because the number of orange IDP checks in Methodology for C++ criterion level 1 is zero.



## Generating Reports of Verification Results

In this section...
“Polyspace Report Generator Overview” on page 4-31
“Generating Verification Reports” on page 4-32

### Polyspace Report Generator Overview

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

The Polyspace Report Generator provides the following report templates:

- **Coding Rules Report** – Provides information about compliance with MISRA C® Coding Rules, as well as Polyspace configuration settings for the verification.
- **Developer Report** – Provides information useful to developers, including summary results, detailed lists of red, orange, and gray checks, and Polyspace configuration settings for the verification. Detailed results are sorted by type of check (Proven Run-Time Violations, Proven Unreachable Code Branches, Unreachable Functions, and Unproven Run-Time Checks).
- **Developer Review Report** – Provides the same information as the Developer Report, but reviewed results are sorted by review classification (High, Medium, Low, Not a defect) and status, and untagged checks are sorted by file location.
- **Developer with Green Checks Report** – Provides the same content as the Developer Report, but also includes a detailed list of green checks.
- **Quality Report** – Provides information useful to quality engineers, including summary results, statistics about the code, graphs showing distributions of checks per file, and Polyspace configuration settings for the verification.
- **Software Quality Objectives Report** – Provides comprehensive information on software quality objectives (SQO), including code metrics, code analysis (coding-rules checker results), code verification (run-time checks), and the configuration settings for the verification. The code

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

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

- HTML
- PDF
- RTF
- WORD
- XML

---

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

---

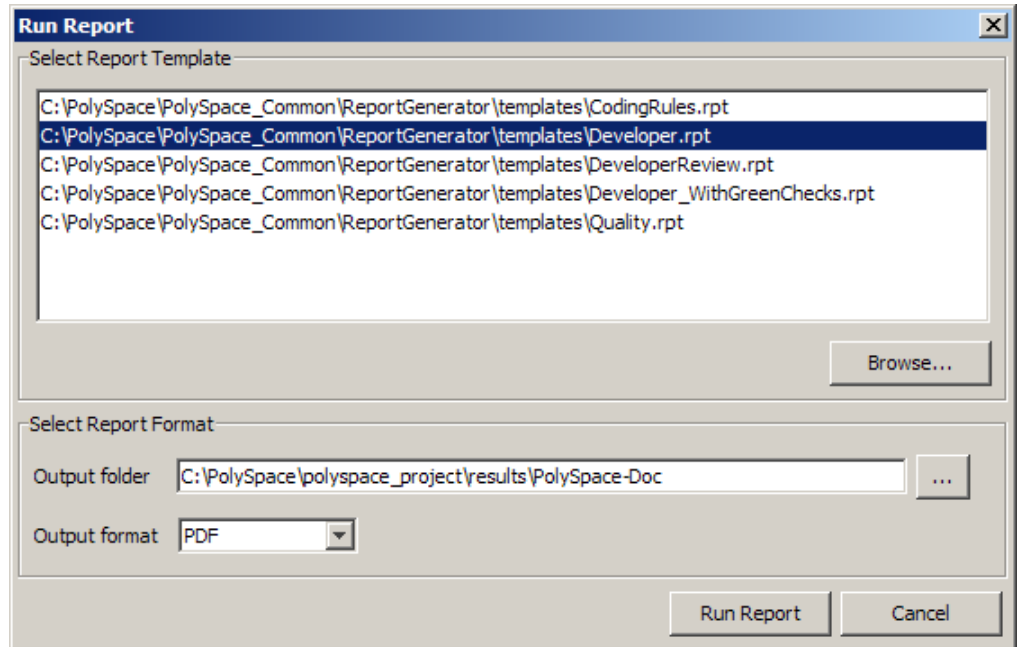
### Generating Verification Reports

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

To generate a verification report:

- 1** If your verification results are not already open, open them.
- 2** Select **Run > Run Report > Run Report**.

The Run Report dialog box opens.



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

The software creates the specified report.



# Checking Compliance with Coding Rules

---

- “About Checking Compliance with Coding Rules Tutorial” on page 5-2
- “Setting Up Coding Rules Checking” on page 5-4
- “Running a Verification with Coding Rules Checking” on page 5-13

## About Checking Compliance with Coding Rules Tutorial

In this section...
“Overview” on page 5-2
“Before You Start” on page 5-3

### Overview

Polyspace software allows you to analyze code to demonstrate compliance with established C++ coding standards (MISRA C++:2008 or JSF++:2005).

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

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

For more information on the coding rules checker, see “Checking Coding Rules” in the *Polyspace Products for C++ User’s Guide*.

In this tutorial, you learn how to:

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

## **Before You Start**

For this tutorial, you check the JSF++ compliance of the file `training.cpp`, using the project that you created in Chapter 2, “Setting Up a Polyspace Project”.

# Setting Up Coding Rules Checking

### In this section...

“Opening Your Project” on page 5-4

“Creating New Verification” on page 5-5

“Setting the JSF++ Checking Option” on page 5-7

“Creating a JSF++ Rules File” on page 5-7

“Excluding Files from JSF++ Checking” on page 5-10

“Configuring Text and XML Editors” on page 5-11

“Saving the Project” on page 5-12

## Opening Your Project

For this tutorial, you modify the project in `Training_Project.cfg` to include JSF++ checking. You use the Project Manager perspective to modify the project.

To open `Training_Project.cfg`:

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

The Open a Polyspace project file dialog box opens.

**2** Navigate to `polyspace_project`.

**3** Select `Training_Project.cfg`.

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




## Creating New Verification

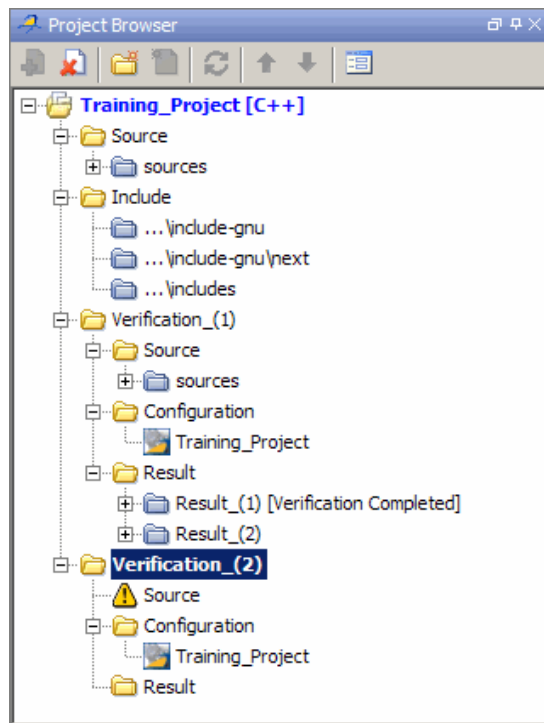
A Polyspace project can contain multiple verifications. Each verification may verify a different set of source files, and may use different analysis options. In this section, you create a second verification to check coding rules compliance for the `training.cpp` file.

To create a new verification in `Training_Project`:

**1** In the Project Browser, select **Training\_Project [C++]**.

**2** Click the Create a new verification icon  in the upper left the Project Browser.

A new verification, `Verification_(2)`, appears in the Project Browser.

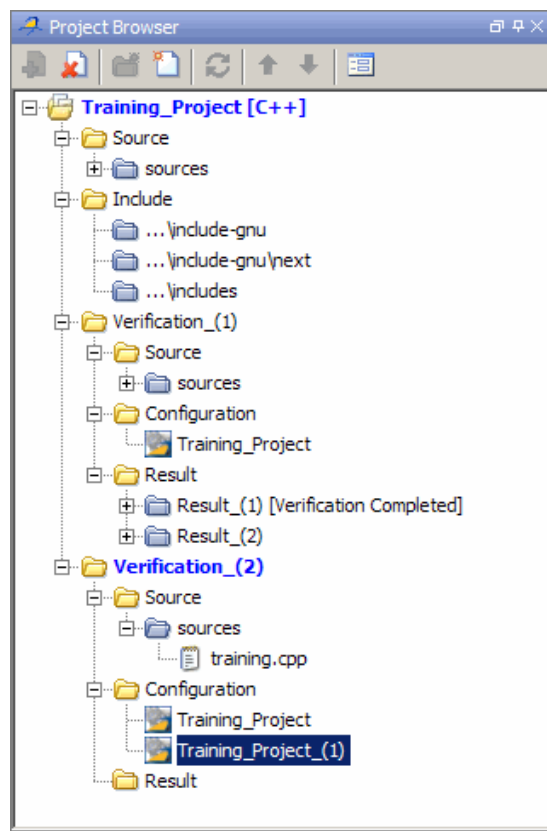


- 3 In the Project Browser Source tree, right-click `training.cpp`, and select **Copy Source File to > Verification\_(2)**.

The `training.cpp` file appears in the Source tree of `Verification_(2)`.

- 4 Right-click the Configuration folder in `Verification_(2)`, and select **Create New Configuration**.

The Project Browser now looks like the following figure.



## Setting the JSF++ Checking Option

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

- 1 Select the **Training\_Project\_(1)** Configuration in the Project Browser.
- 2 In the Analysis options, select **Compliance with standards > Coding rules checker**.

The software displays the JSF C++ rules checker options, `-jsf-coding-rules` and `-includes-to-ignore`.

<input type="checkbox"/> Coding rules checker			
<input checked="" type="checkbox"/> Check JSF C++ rules	<input checked="" type="checkbox"/>		
... JSF C++ rules configuration		...	-jsf-coding-rules
<input type="checkbox"/> Check MISRA C++ rules	<input type="checkbox"/>		
... MISRA C++ rules configuration		...	-misra-cpp
... Files and folders to ignore		...	-includes-to-ignore

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

- 3 Select the **Check JSF C++: rules** check box.


## Creating a JSF++ Rules File

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

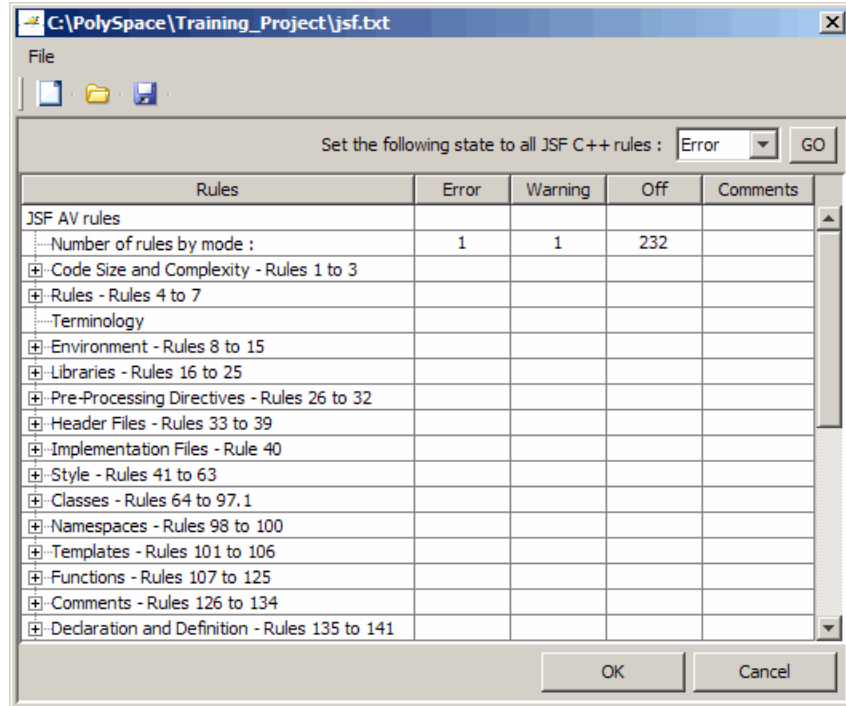
- “Opening a New Rules File” on page 5-7
- “Setting All the Rules to Off” on page 5-9
- “Selecting the Rules to Check ” on page 5-9

## Opening a New Rules File

To create a new rules file:

- 1 Click the browse button  to the right of the **JSF C++ rules configuration** option.

A table of rules appears.



2 For each rule, you can specify one of the following states.

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

---

## Setting All the Rules to Off

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

To set the state of all rules to **Off**:

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

## Selecting the Rules to Check

To select the rules to check for this tutorial:

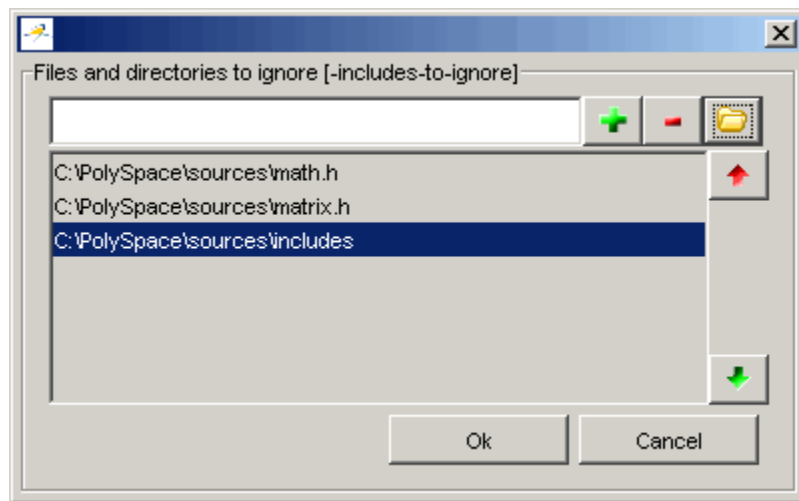
- 1 Expand the set of rules named **Type Conversions - Rules 177 to 185**.
- 2 Select the **Warning** column for rule 180.
- 3 Expand the set of rules names **Flow Control Structures - Rules 186 to 201**.
- 4 Select the **Error** column for rule 191.
- 5 Click **OK** to save the rules and close the window.  
The **Save as** dialog box opens.
- 6 In **File**, enter `jsf.txt`
- 7 Click **OK** to save the file and close the dialog box.

## Excluding Files from JSF++ Checking

You can exclude files from JSF++ checking. You might want to exclude some included files. To exclude `math.h` from the JSF++ checking of the project `Training_Project.cfg`:

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

The Files and folders to ignore (includes-to-ignore) dialog box opens.



- 2 Click the folder icon.

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

- 3 Navigate to the `polyspace_project` folder.

- 4 Select the `includes` folder.

- 5 Click **OK**.

The `includes` folder appears in the list of files to ignore.

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

## Configuring Text and XML Editors

Before you check JSF++ rules, you should configure your text and XML editors in the Preferences. Configuring text and XML editors allows you to view source files and JSF reports directly from the Coding Rules perspective.

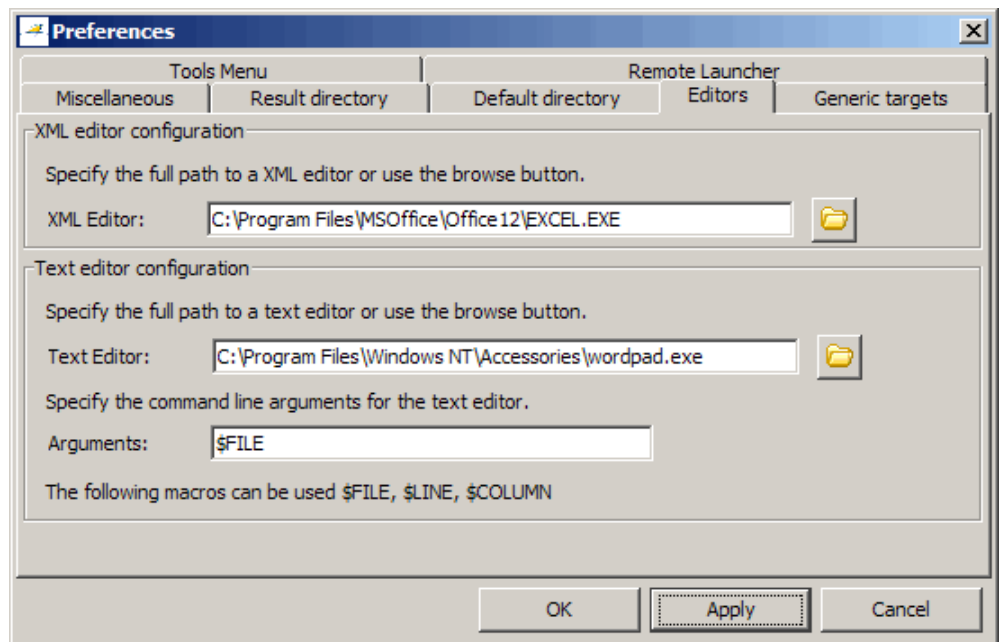
To configure your text and .XML editors:

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

The Preferences dialog box opens.

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

The Editors tab opens.



**3** Specify an XML editor to use to view JSF++ 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**

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



## Running a Verification with Coding Rules Checking

In this section...
“Starting the Verification” on page 5-13
“Examining JSF C++ Violations” on page 5-15
“Opening JSF Report” on page 5-17

### Starting the Verification

When you run a verification with the **Check JSF C++ rules** option selected, the verification checks most of the JSF++ rules during the compile phase. If there is a violation of a rule with state **Error**, the verification stops.

---


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

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

The verification fails because of JSF++ violations. The message “**Verification Failed**” appears at the bottom of the Project Manager perspective, and the Output Summary indicates that the verification has detected JSF errors.

## 5 Checking Compliance with Coding Rules

Output Summary - [C:\PolySpace\Training\_Project\Verification\_(2)\Result\_(1)]

Search:

Class	Description	File	Line	Col
	Training_Project for C++ verification start at Jun 29, 2010 17:38:16			
	Verifier has detected JSF error(s) in the code.			
	Exiting because of previous error			

Detail:

Error:  
Verifier has detected JSF error(s) in the code.

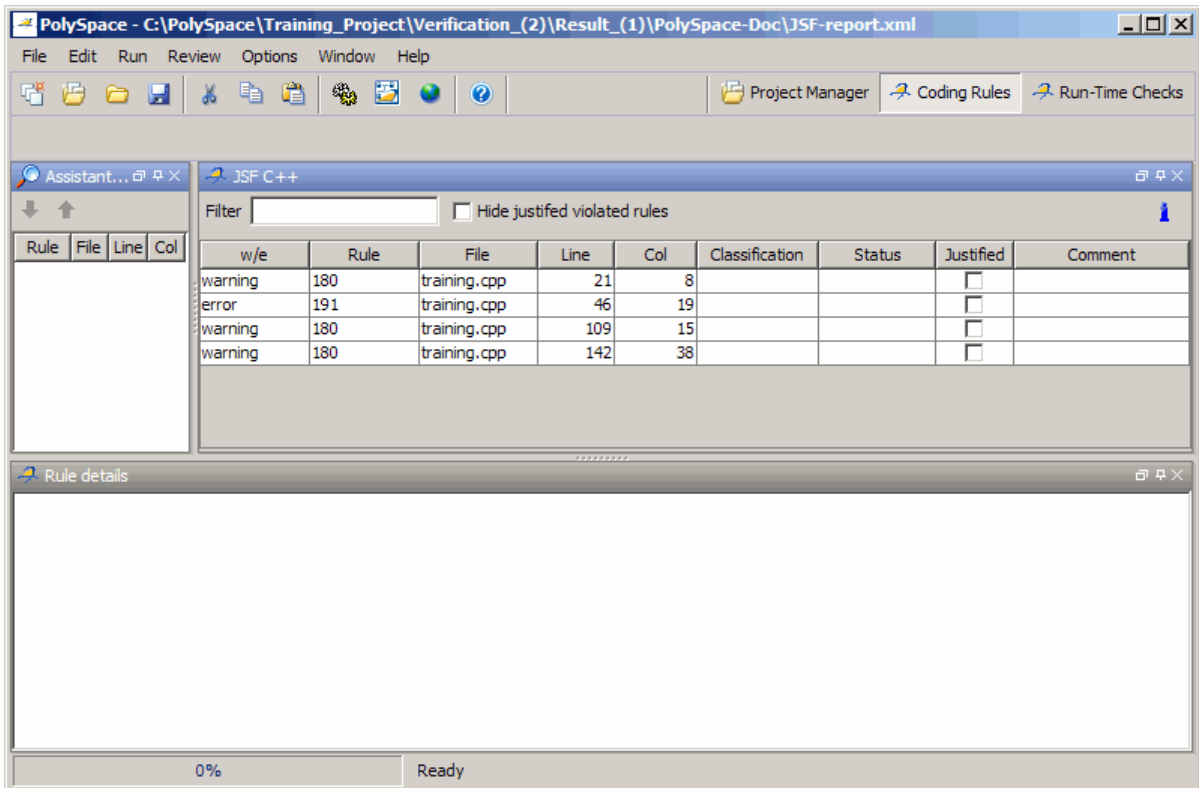
Full Log | Verification Statistics | Progress Monitor | Output Summary

## Examining JSF C++ Violations

To examine the JSF++ violations:

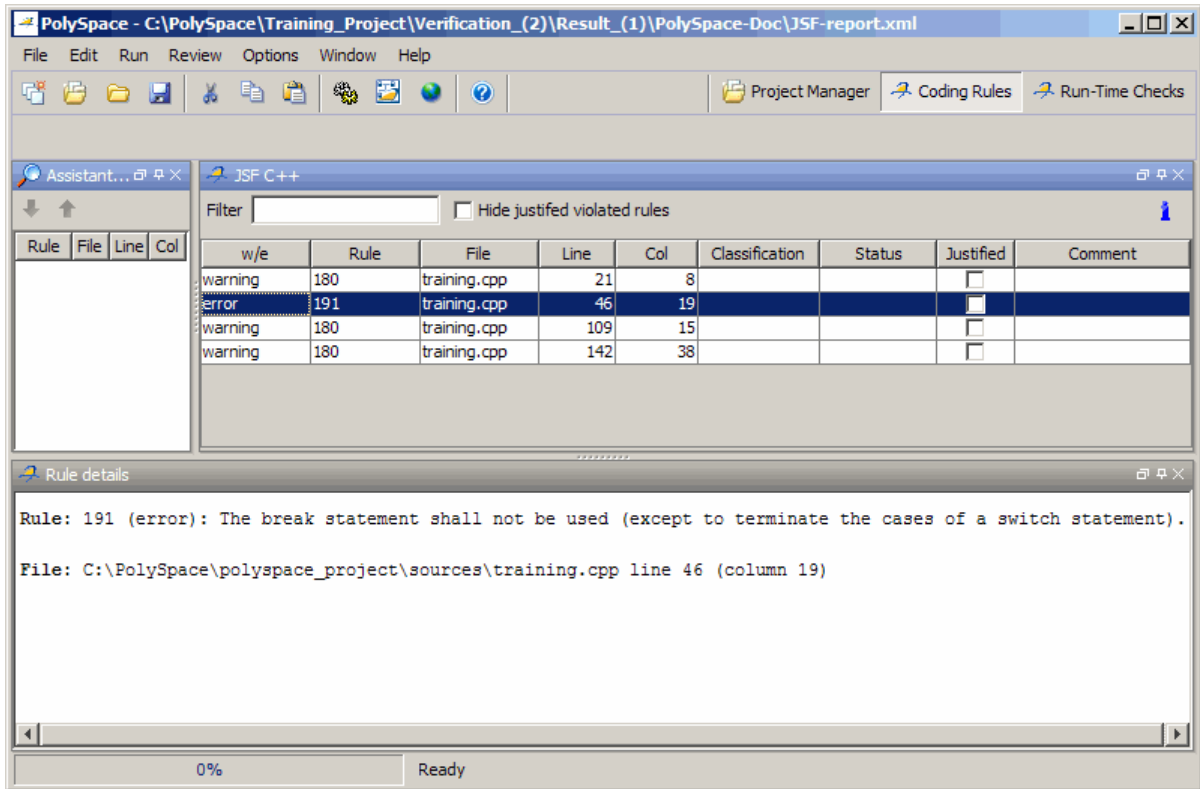
- 1 Double-click **JSF-report.xml** in the Project Browser Result folder.

The Coding Rules perspective appears, displaying a list of JSF++ violations.



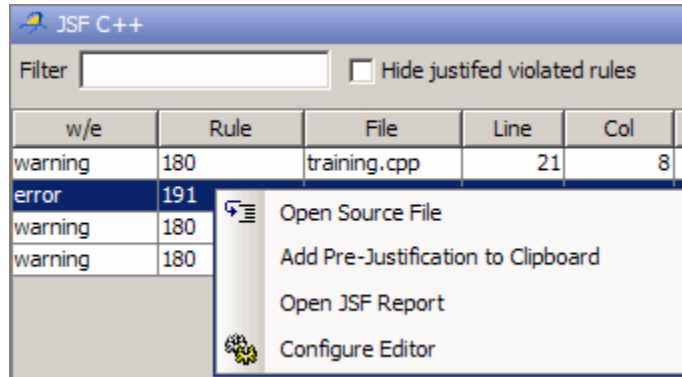
- 2 Click on any of the violations to see a description of the violated rule, the full path of the file in which the violation was found, and the source code containing the violation.

## 5 Checking Compliance with Coding Rules



The log reports a violation of rule 191. A break statement is used in training.cpp.

- 3 Right click the row containing the violation of rule 191 , then select Open Source File.



The training.cpp 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 5-11.

---

#### 4 Correct the JSF++ 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 JSF Report

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

---

**Note** You must configure an XML editor before you can open a JSF report. See “Configuring Text and XML Editors” on page 5-11..

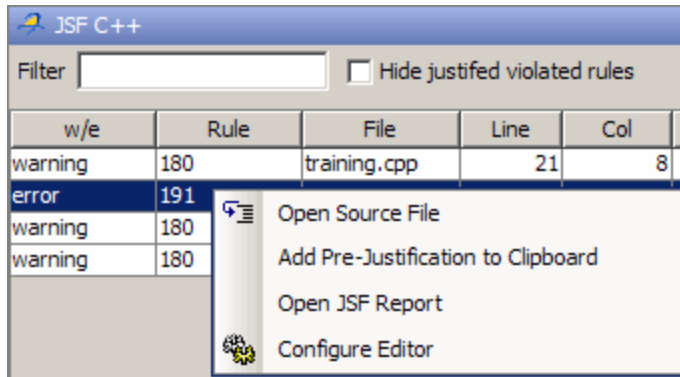
---

To view the JSF report:

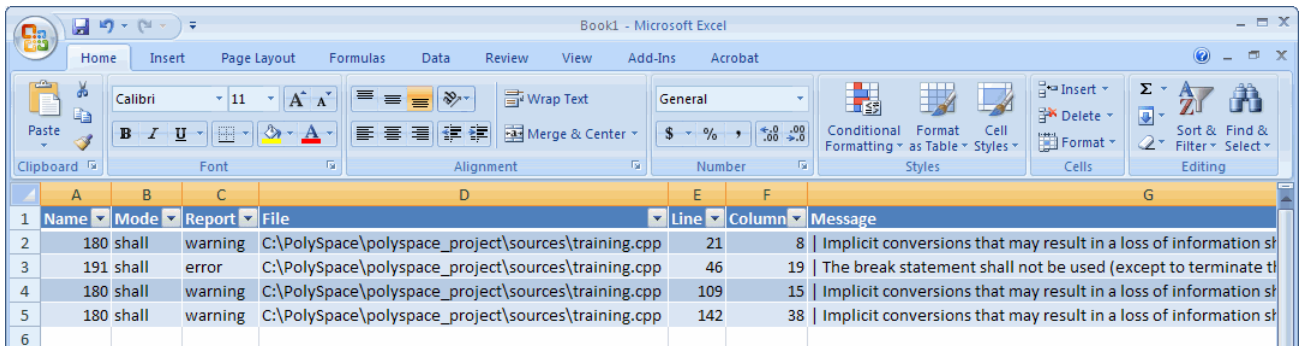
#### 1 Click the **Coding Rules** button in the Polyspace Verification Environment toolbar.

A list of JSF++ violations appears in the Coding Rules perspective.

2 Right click any row in the log, and select **Open JSF Report**.



The report opens in your XML editor.



## A

- active project
  - definition 3-21
  - setting 3-21
- analysis options 2-12
  - JSF++ compliance 5-7
- ANSI compliance 3-10
- assistant mode
  - criterion 4-25
  - custom methodology 4-29
  - methodology 4-25
  - methodology for C++ 4-25
  - overview 4-24
  - reviewing checks 4-27
  - selection 4-24
  - use 4-24 4-27

## C

- call graph 4-11
- call tree view 4-4
- calling sequence 4-11
- cfg. *See* configuration file
- client 1-5 3-2
  - installation 1-12
  - verification on 3-28
- coding review progress view 4-4 4-12
- Coding Rules perspective 1-5
- color-coding of verification results 1-2 to 1-3 4-7
- compile log
  - Launcher 3-30
  - Project Manager 3-12
  - Spooler 3-13
- compile phase 3-10
- compliance
  - ANSI 3-10
  - JSF C++ 5-1
  - MISRA C 1-3
- configuration file
  - definition 2-3

- custom methodology
  - definition 4-29

## D

- desktop file
  - definition 2-3
- division by zero
  - example 4-18
- downloading
  - results 3-18
- dsk. *See* desktop file

## E

- expert mode
  - filters 4-19
  - use 4-9

## F

- files
  - includes 2-10
  - source 2-10
- filters 4-19
- folders
  - includes 2-10
  - sources 2-10

## H

- hardware requirements 3-19
- help
  - accessing 1-16

## I

- installation
  - Polyspace Client for C/C++ 1-12
  - Polyspace products 1-12
  - Polyspace Server for C/C++ 1-12

**J**

- JSF++ compliance
  - analysis option 5-7
  - checking 5-1
  - file exclusion 5-10
  - log 5-15
  - rules file 5-7

**L**

- Launcher
  - monitoring verification progress 3-30
  - viewing logs 3-30
  - window
    - progress bar 3-30
- licenses
  - obtaining 1-12
- logs
  - compile
    - Launcher 3-30
    - Project Manager 3-12
    - Spooler 3-13
  - full
    - Launcher 3-30
    - Project Manager 3-12
    - Spooler 3-13
  - stats
    - Launcher 3-30
    - Project Manager 3-12
    - Spooler 3-13
  - viewing
    - Launcher 3-30
    - Project Manager 3-12
    - Spooler 3-13

**M**

- manual mode
  - overview 4-9
  - selection 4-9

- methodology for C++ 4-25
- MISRA C compliance 1-3

**P**

- Polyspace Client for C/C++
  - installation 1-12
  - license 1-12
- Polyspace In One Click
  - active project 3-21
  - overview 3-21
  - sending files to Polyspace software 3-23
  - starting verification 3-23
  - use 3-21
- Polyspace products for C++
  - components 1-5
  - installation 1-12
  - licenses 1-12
  - overview 1-2
  - related products 1-17
  - user interface 1-5
  - workflow 1-13
- Polyspace Queue Manager Interface. *See* Spooler
- Polyspace Server for C/C++
  - installation 1-12
  - license 1-12
- Polyspace verification environment
  - opening 2-5
- ppm. *See* Polyspace project model file
- preferences
  - Launcher
    - server detection 3-20
  - Project Manager
    - default server mode 3-10
  - Viewer
    - assistant configuration 4-25
- procedural entities view 4-4
- product overview 1-2
- progress bar
  - Launcher window 3-30



- Project Manager window 3-12
- project
  - creation 2-3 2-7
  - definition 2-3
  - file types
    - configuration file 2-3
    - desktop file 2-3
    - Polyspace project model file 2-3
  - folders
    - includes 2-4
    - results 2-4
    - sources 2-4
  - opening 3-4
  - saving 2-15
- Project Manager
  - monitoring verification progress 3-12
  - opening 2-5
  - overview 2-5
  - perspective 2-5
  - starting verification on client 3-28
  - starting verification on server 3-10
  - viewing logs 3-12
  - window
    - progress bar 3-12
- Project Manager perspective 1-5
- project model file. *See* Polyspace project model file

**R**

- related products 1-17
  - Polyspace products for linking to Models 1-17
  - Polyspace products for verifying Ada code 1-17
  - Polyspace products for verifying C code 1-17
- reports
  - generation 4-31
- results
  - downloading from server 3-18
  - opening 4-3

- report generation 4-31
- reviewing 4-1
- rte view. *See* procedural entities view
- Run-time checks perspective
  - call tree view 4-4
  - coding review progress view 4-4
  - procedural entities view 4-4
  - selected check view 4-4
  - source code view 4-4
  - variables view 4-4
- Run-Time Checks perspective 1-5
  - opening 4-3

**S**

- selected check view 4-4
- server 1-5 3-2
  - detection 3-20
  - information in preferences 3-20
  - installation 1-12 3-20
  - verification on 3-10
- source code view 4-4
- Spooler 1-5
  - monitoring verification progress 3-13
  - removing verification from queue 3-18
  - use 3-13
  - viewing log 3-13

**T**

- target environment 2-11
- troubleshooting failed verification 3-19

**U**

- unreachable code
  - example 4-15

**V**

- variables view 4-4

## verification

- Ada code 1-17
- C code 1-17
- C++ code 1-2
- client 3-2
- compile phase 3-10
- failed 3-19
- monitoring progress
  - Launcher 3-30
  - Project Manager 3-12
  - Spooler 3-13
- phases 3-10
- results
  - color-coding 1-2 to 1-3
  - opening 4-3
  - report generation 4-31
  - reviewing 4-1
- running 3-2
- running on client 3-28

- running on server 3-10

## starting

- from Launcher 3-2
- from Polyspace In One Click 3-2 3-23
- from Project Manager 3-10 3-28

## stopping 3-32

- troubleshooting 3-19
- with JSF++ checking 5-13

## Verification

- stopping 3-31

## Viewer

- window
  - overview 4-4

**W**

## workflow

- basic 1-13
- in this guide 1-14