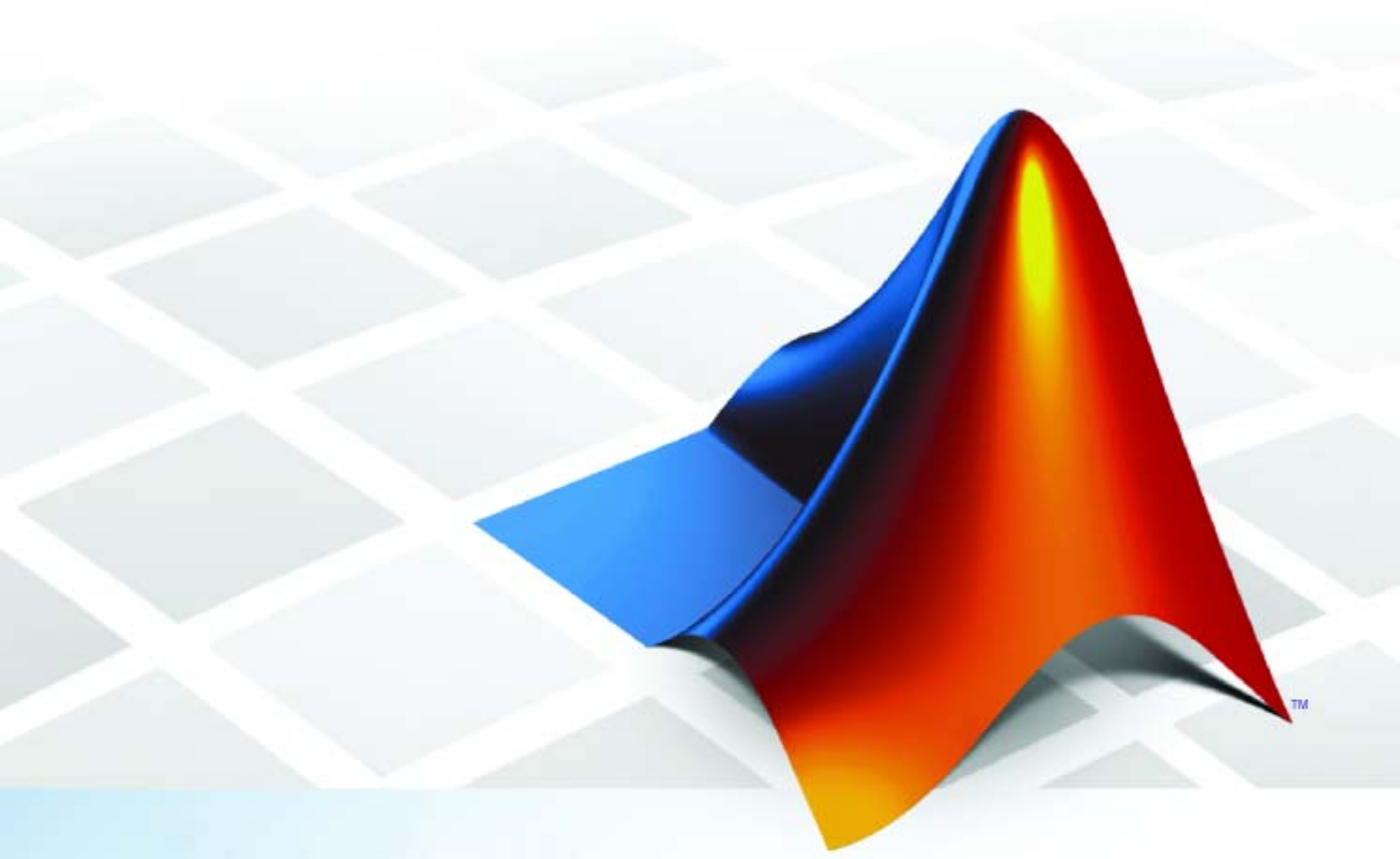
PolySpace® Products for C++ 7

Getting Started Guide



The MathWorks™

Accelerating the pace of engineering and science

How to Contact The MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

PolySpace® Products for C++ Getting Started Guide

Trademarks

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

Patents

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

Revision History

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

Contents

1 Introduction to PolySpace Products for Verifying C++ Code

2 Setting Up a Project File

3 Running a Verification

4 Reviewing Verification Results

5 Checking Compliance with Coding Rules

6 Using a PolySpace Project Model File

Index

1

Introduction to PolySpace Products for Verifying C++ Code

Product Overview

In this section...

Ensures Software Reliability

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

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

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

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

Decreases Development Time

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

You use PolySpace software to verify C++ source code prior to compilation. To verify the source code, you set up verification parameters in a project, run

the verification, and review the results. This process takes significantly less time than using manual methods or using tools that require you to modify code or run test cases.

A graphical user interface helps you to efficiently review verification results. Results are color-coded:

- Green indicates code that never has an error.
- Red indicates code that always has an error.
- Gray indicates unreachable code (dead code).
- Orange indicates unproven code (code that might have an error).

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

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

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

Improves the Development Process

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

PolySpace verification software supports code verification throughout the development process:

- An individual developer can find and fix run-time errors during the initial coding phase.
- Quality assurance can check overall reliability of an application.

- Managers can monitor application reliability by generating reports from the verification results.

Product Components

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

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

The user interface includes:

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

Installing PolySpace Products

In this section...

Finding the Installation Instructions

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

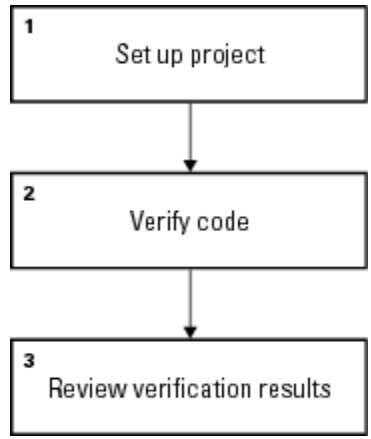
See "PolySpace License Installation" in the *PolySpace Installation Guide* for information about obtaining and installing licenses for PolySpace products.

Working with PolySpace Software

In this section...

Basic Workflow

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



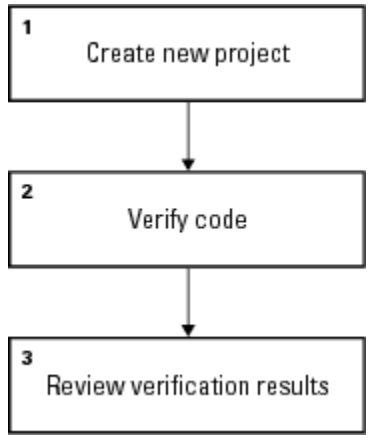
In this workflow, you:

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

   You can use the Launcher to start the verification or you can select files from a Microsoft® Windows® folder and send them to thePolySpace 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 Viewer to review verification results.

The Workflow in This Guide

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



In this workflow, you will:

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

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

2. Verify a single 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:

   - Use the Launcher to start a verification that runs on a server.
   - Use PolySpace In One Click to start a verification that runs on a server.
   - Use the Launcher to start a verification that runs on a client.

3. Review the verification results.

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

Working with PolySpace Project Model Files

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

Learning More

In this section...

Product Help

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

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

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

The MathWorks Online

For additional information and support, see:

```
www.mathworks.com/products/polyspace
```

Related Products

In this section...

PolySpace Products for Verifying C Code

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/polyspacemodelsl/`

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

2

Setting Up a Project File

About This Tutorial

In this section...

Overview

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

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?

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

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

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

A project file has one of the following file types:

Project Type	File Extension	Description
Configuration	`cfg`	Required for running a verification. Does not include generic target processors.
PolySpace Project Model	`ppm`	Used to populate a project with analysis options, including generic target processors.

Project Type	File Extension	Description
Desktop	`dsk`	Obsolete. Used in earlier versions of PolySpace software for running a verification on a client computer.

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

Preparing the Project Folders

Before you start verifying 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, include files, and results within the project folder.

For this tutorial, prepare a project folder as follows:

1. Create a project folder named `polyspace_project`.
2. Open `polyspace_project`, and create the following folders:
   - `sources`
   - `includes`
   - `results`
3. Copy the file `training.cpp` from

   `Install_folder``\Examples\Demo_Cpp_Long\sources`

   to

   `polyspace_project\sources`

   where *`Install_folder`* is the installation folder.

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

`Install_folder``\Examples\Demo_Cpp_Long\sources`

to

`polyspace_project\includes`.

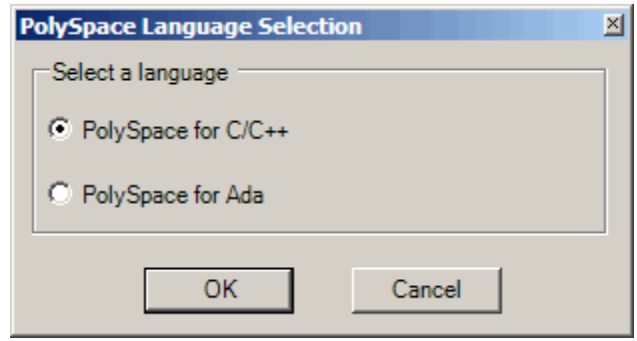
Opening the PolySpace Launcher

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

To open the PolySpace Launcher:

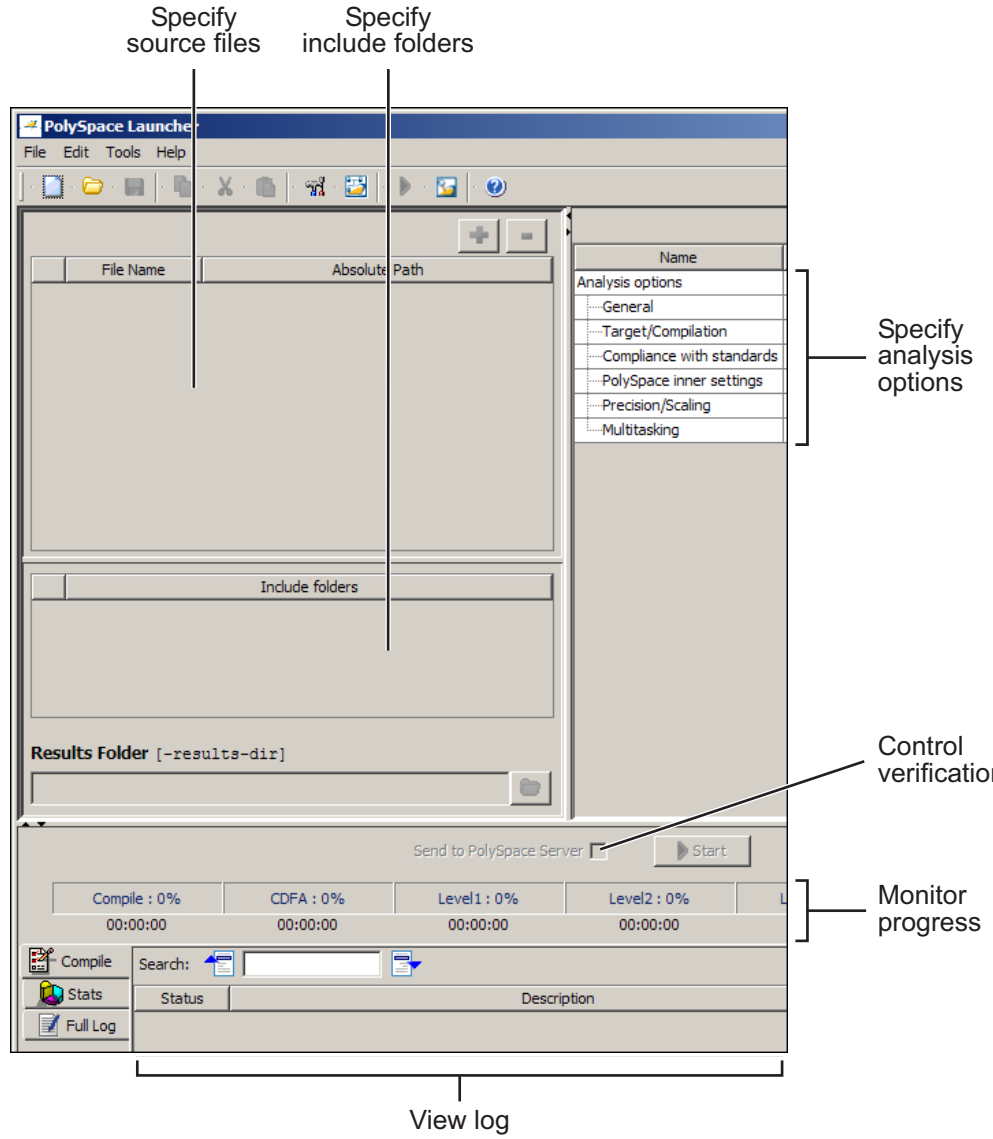
- Double-click the PolySpace Launcher icon on your desktop.

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



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

The PolySpace Launcher window opens.



The Launcher window has three main sections.

Use this section...	For...
Upper-left	Specifying: • Source files • Include folders • Results folder
Upper-right	Specifying analysis options
Lower	Controlling and monitoring a verification

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

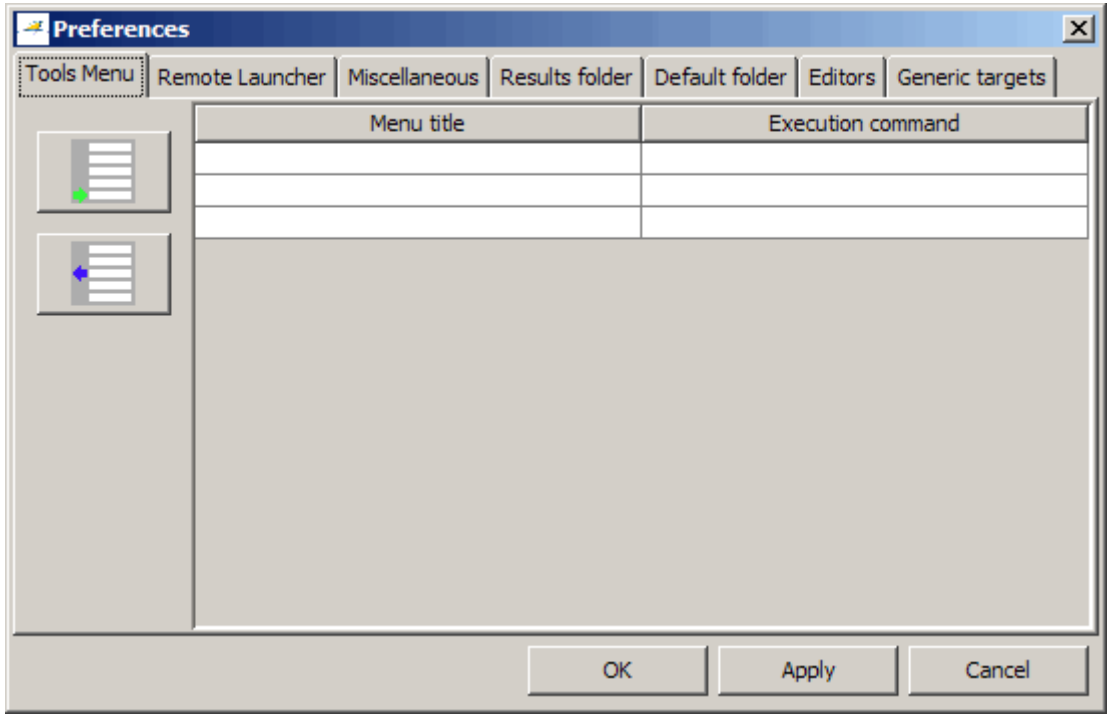
Changing the Default Folder

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

To change the default folder to the project folder:

1 Select **Edit > Preferences**.

The **Preferences** dialog box appears.

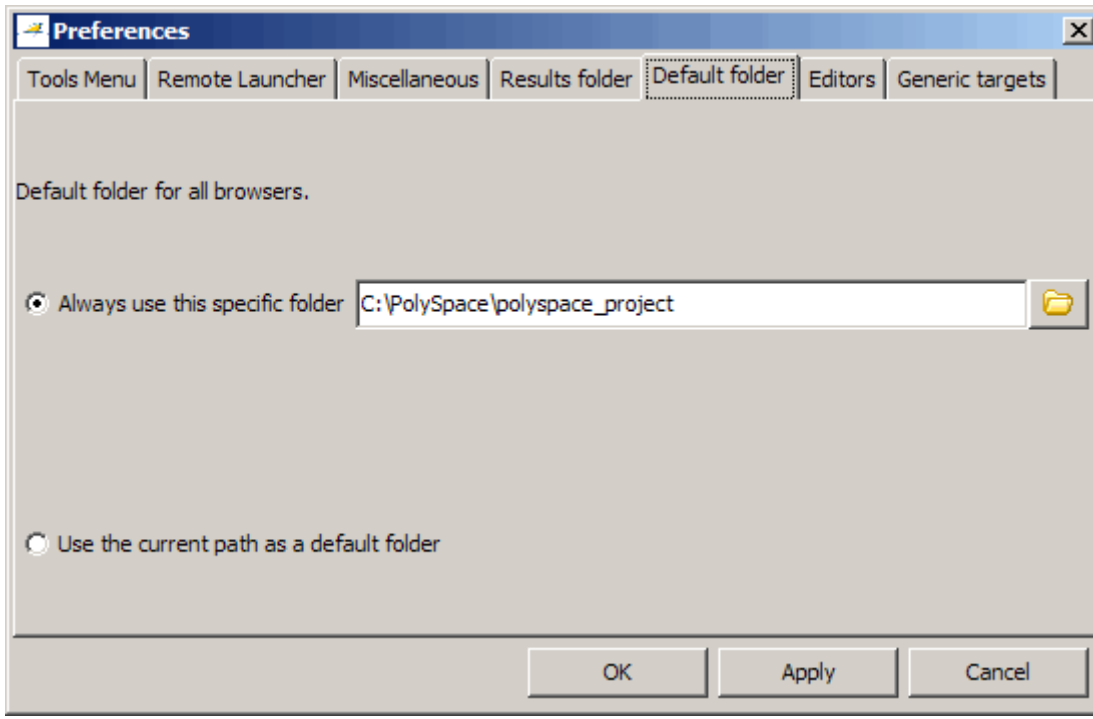


2 Select the **Default folder** tab.

3 Select **Always use this specific folder** if it is not already selected.

4 Enter or navigate to the project folder that you created earlier. In this example, the project folder is `C:\PolySpace\polyspace_project`.

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



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

Creating a New Project to Verify 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:

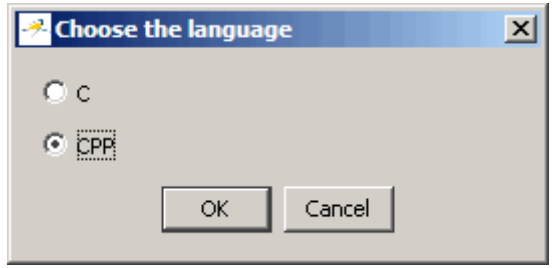
- "Opening a New project" on page 2-10
- "Specifying the Source Files, Include Folders, and Results Folder" on page 2-11
- "Specifying the Analysis Options" on page 2-14
- "Saving the Project" on page 2-17

Opening a New project

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

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

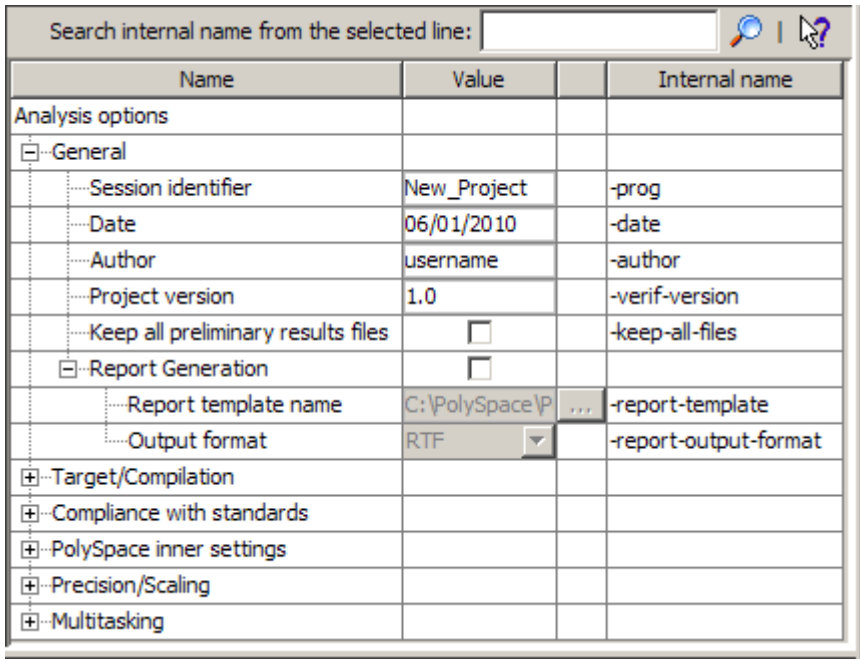
   The **Choose the language** dialog box appears:



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

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

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



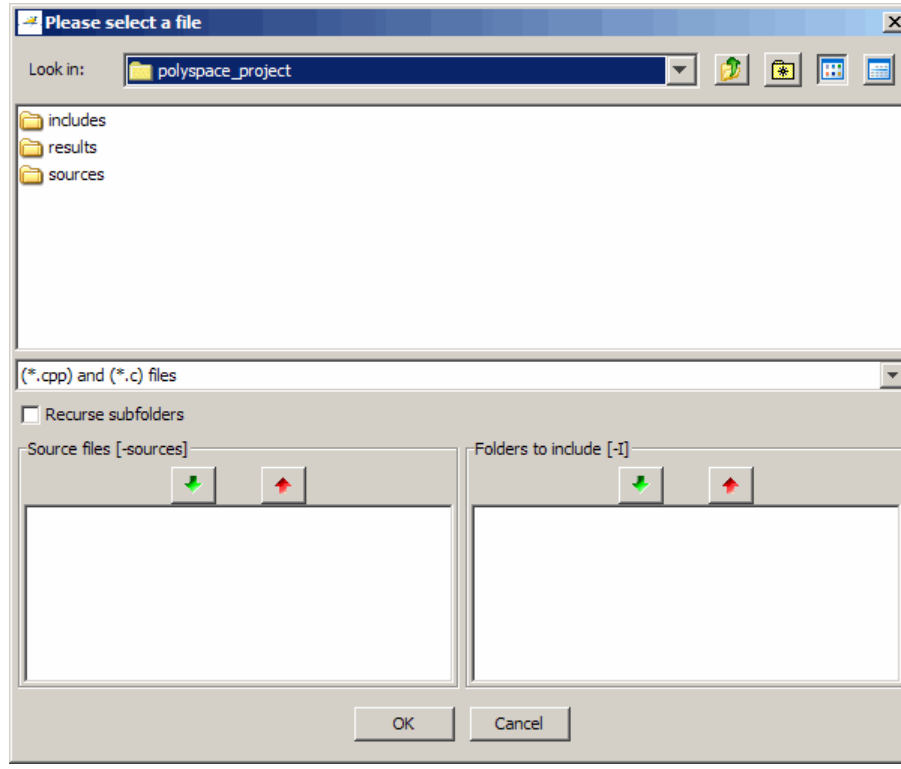
Specifying the Source Files, Include Folders, and Results Folder

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

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

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



2 The project folder `polyspace_project` should appear in the **Look in** drop-down box. If it does not, navigate to that folder.

3 Double-click the `sources` folder.

4 Select the file `training.cpp` and then click the green down arrow button in the **Source files** section.

The path for `training.cpp` appears in the source files list.

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

5 Navigate back to the `polyspace_project` folder.

Select the folder `includes`, then click the green down arrow button in the **Folders to include** section.

The path for the folder appears in the list of folders to include.

6 Navigate to the folder *`Install_folder`*`\Verifier\include`.

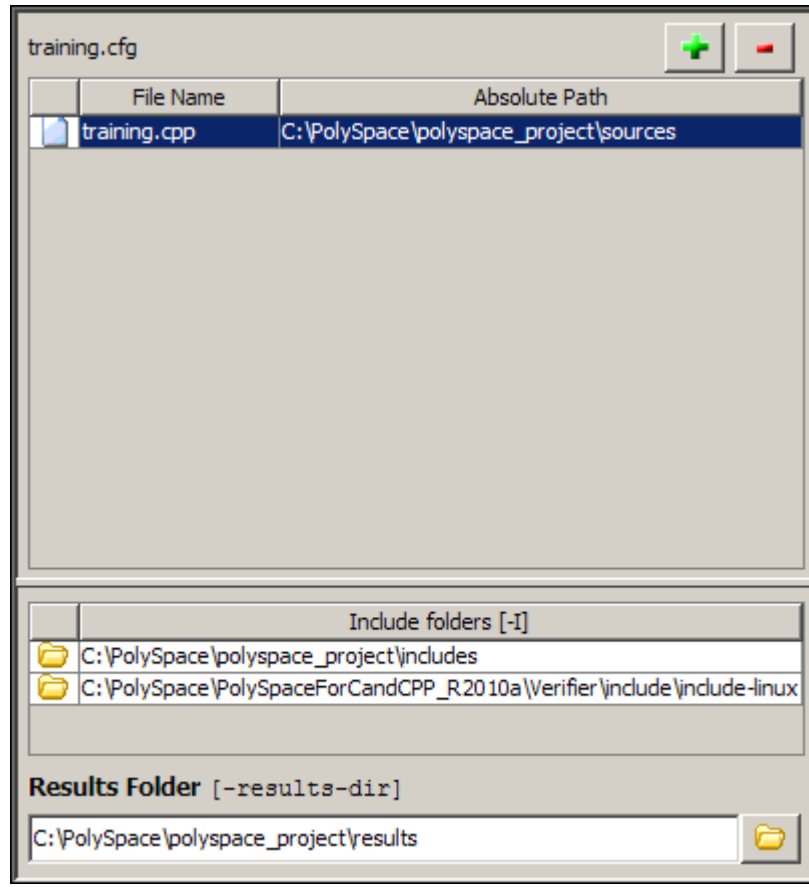
7 Select the folder `include-linux`, then click the green down arrow button in the **Folders to include** section.

Note This tutorial uses a Linux OS target, therefore you must include the Linux library files. When verifying your code, you should include the standard headers for your compiler.

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

9 In **Results Folder**, specify the folder for the verification results. Enter the path for the results folder that you created earlier. In this example, the results folder is `C:\PolySpace\polyspace_project\results`.

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



Specifying the Analysis Options

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

To specify the analysis options for this tutorial:

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

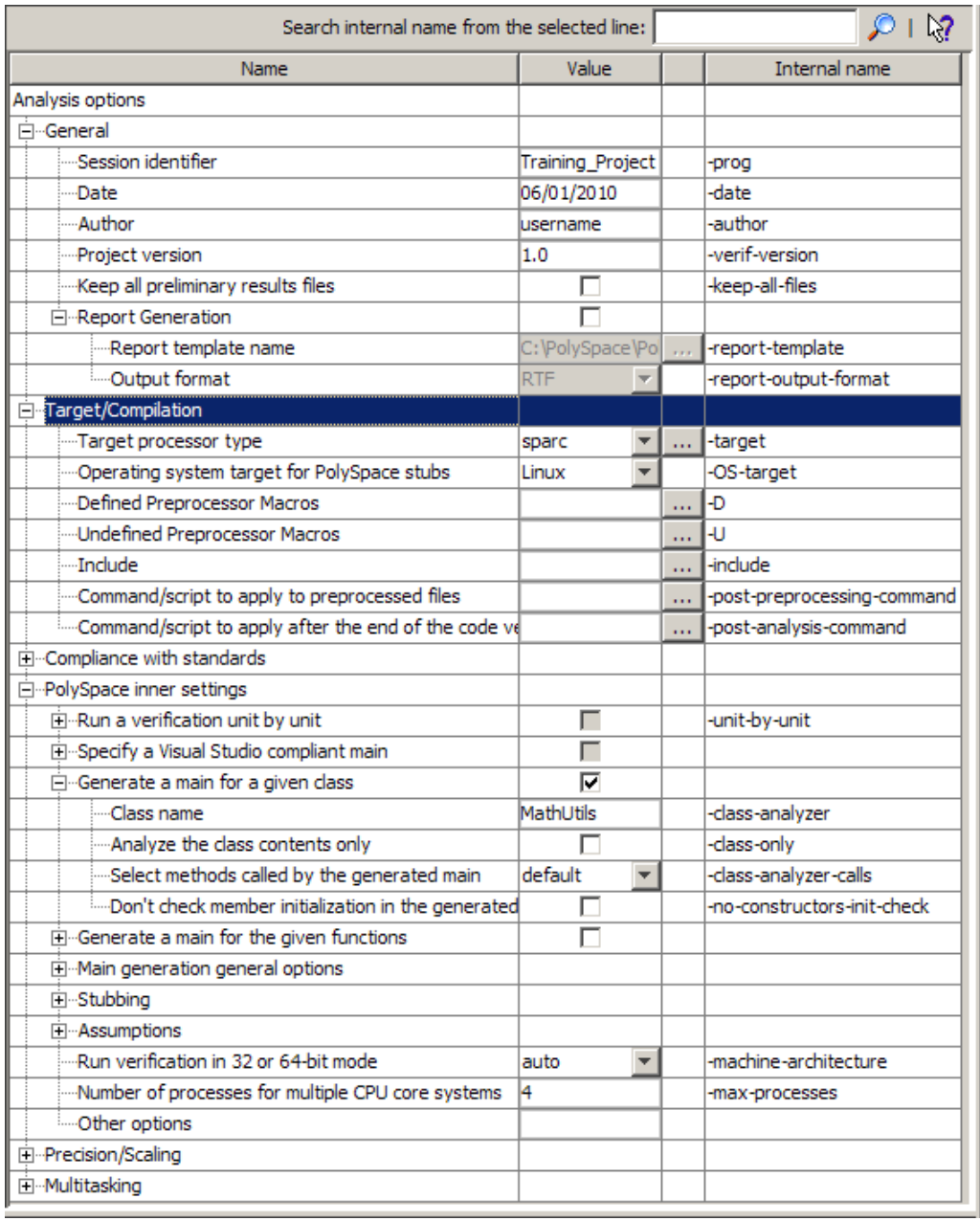
   Note The session identifier cannot contain spaces.

2. Expand the **PolySpace inner settings** section and select the **Generate a main using a given class** check box. This enables the `-class-analyzer` option and allows you to specify the class you want to verify. Expand the **Generate a main using a given class** section and type in MathUtils as the class name.

3. Expand the **Target/Compilation** section. Because you included Linux header files for this project, you must select a Linux® OS target. This will provide PolySpace with a set of predefined compilation flags that are known to be default or implicit compile options for the target OS. Select **Linux** from the drop-down menu next to **Operating system target for PolySpace stubs**.

4. Keep the default values for all other options.

The analysis options will now look like this.



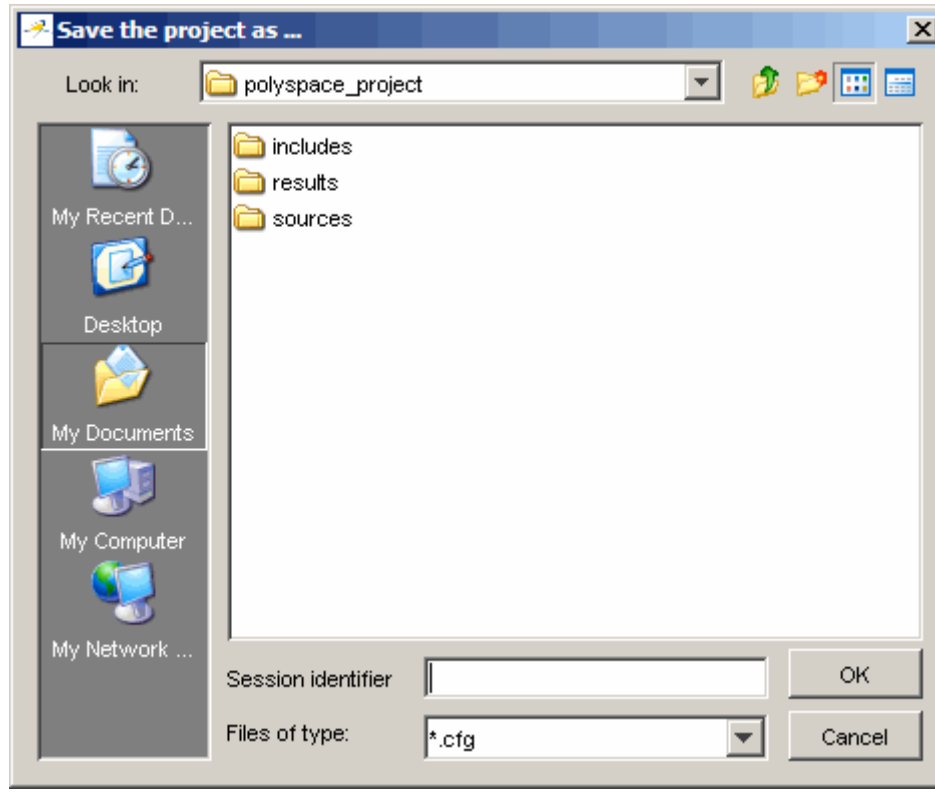
Name	Value		Internal name
Analysis options			
General			
Session identifier	Training_Project		-prog
Date	06/01/2010		-date
Author	username		-author
Project version	1.0		-verif-version
Keep all preliminary results files	☐		-keep-all-files
Report Generation	☐		
Report template name	C:\PolySpace\Po	...	-report-template
Output format	RTF		-report-output-format
Target/Compilation			
Target processor type	sparc	...	-target
Operating system target for PolySpace stubs	Linux		-OS-target
Defined Preprocessor Macros		...	-D
Undefined Preprocessor Macros		...	-U
Include		...	-include
Command/script to apply to preprocessed files		...	-post-preprocessing-command
Command/script to apply after the end of the code ve		...	-post-analysis-command
Compliance with standards			
PolySpace inner settings			
Run a verification unit by unit	☐		-unit-by-unit
Specify a Visual Studio compliant main	☐		
Generate a main for a given class	☑		
Class name	MathUtils		-class-analyzer
Analyze the class contents only	☐		-class-only
Select methods called by the generated main	default		-class-analyzer-calls
Don't check member initialization in the generated	☐		-no-constructors-init-check
Generate a main for the given functions	☐		
Main generation general options			
Stubbing			
Assumptions			
Run verification in 32 or 64-bit mode	auto		-machine-architecture
Number of processes for multiple CPU core systems	4		-max-processes
Other options			
Precision/Scaling			
Multitasking			

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:

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



2 In **Look in**, leave the default folder, `polyspace_project`.

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

4 In **Files of type**, leave the default `*.cfg`. You must have a project file with type `cfg` to run a verification.

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

---

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

3

Running a Verification

About This Tutorial

In this section...

Overview

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

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

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

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

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

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

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

Each verification stores the same results in `polyspace_project\results`. You review these results in the tutorial Chapter 4, "Reviewing Verification Results".

Before You Start

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

Opening the Project

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

To open `training.cfg`:

1. If the PolySpace Launcher is not already open, open it by double-clicking the PolySpace Launcher icon.
2. Select **File > Open project**.

   The **Please select a file** dialog box opens.
3. In **Look in**, navigate to `polyspace_project`.
4. Select `training.cfg`.
5. Click **Open** to open the file and close the dialog box.

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

In this section...

Starting the Verification

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

To start a verification that runs on a server:

1. Select the **Send to PolySpace Server** check box next to the **Start** button in the middle of the Launcher window.

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

2. Click **Start**.

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

The verification has three main phases:

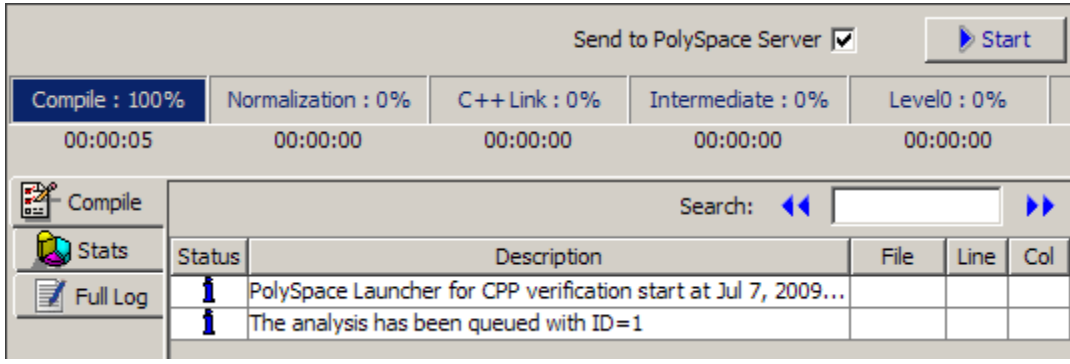
a Checking syntax and semantics (the compile phase). Because PolySpace software is independent of any particular C++ compiler, it ensures that your code is portable, maintainable, and complies with ANSI® standards.

b Generating a main if it does not find a main and the **Generate a Main** option is selected. For more information about generating a main, see "Generate a Main Using a Given Class" 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 finishes:

- A message dialog box tells you that the verification is completed. This message means that the part of the verification that takes place on the client is complete. The rest of the verification runs on the server.
- A message in the log area tells you that the verification was transferred to the server and gives you the identification number (Analysis ID) for the verification. For this verification, the identification number is 2.



3 When you see the message `Verification process completed`, click **OK** to close the message dialog box.

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

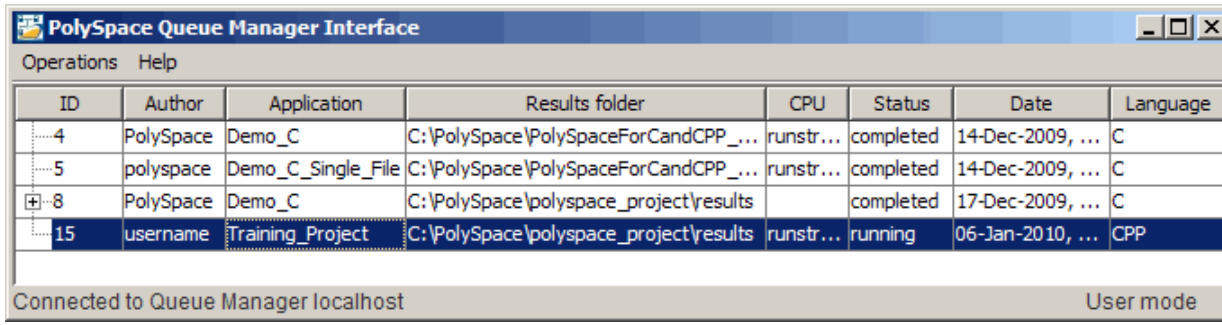
Monitoring the Progress of the Verification

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

To monitor the verification of `Example_Project`:

1. Double-click the **PolySpace Spooler** icon:

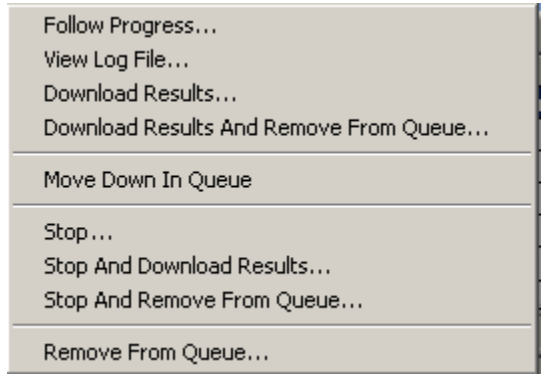
   The **PolySpace Queue Manager Interface** opens.



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

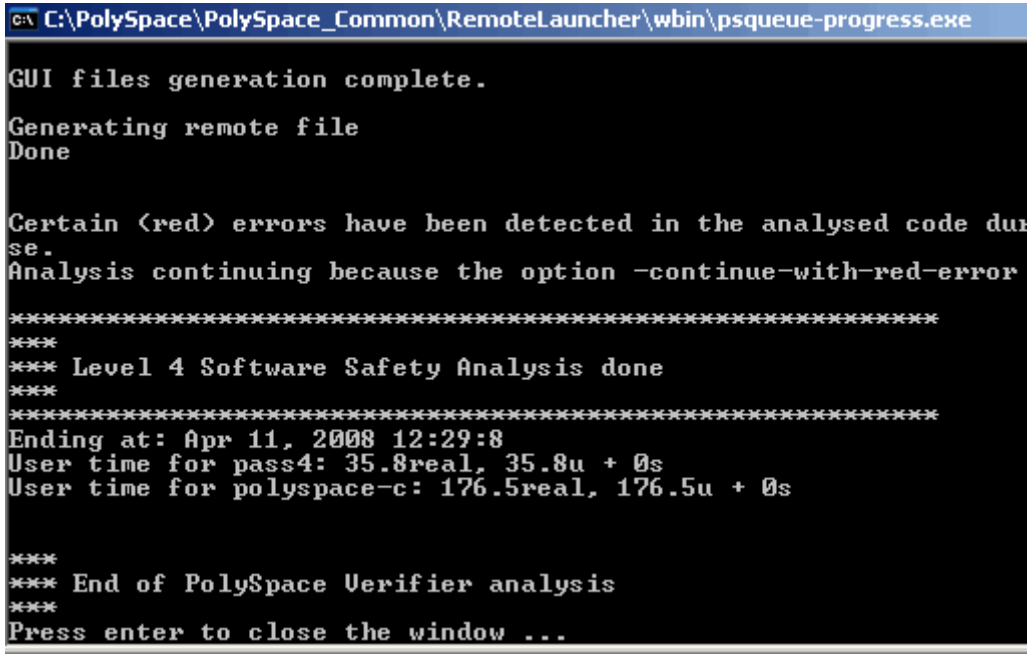
2. Point anywhere in the row for ID 1.

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



4 Select **View log file**.

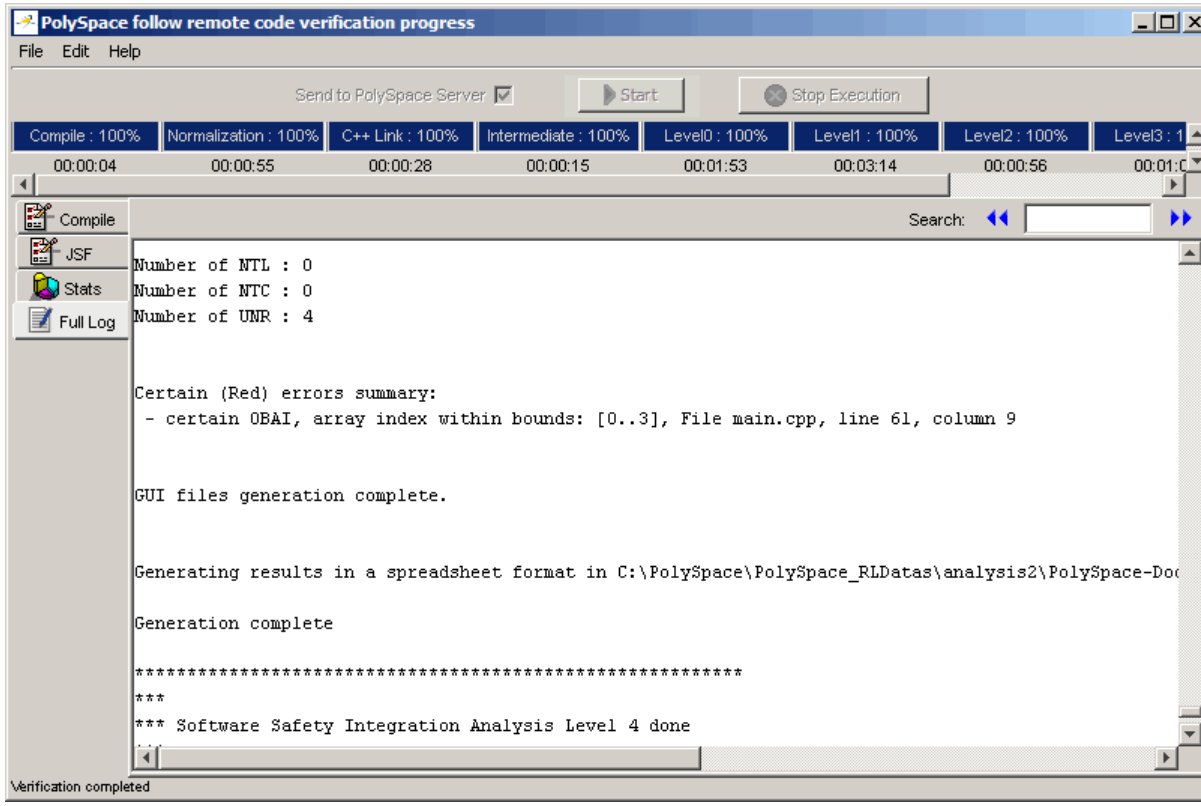
A window opens displaying the last one-hundred lines of the verification.



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

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

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



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

The logs report additional information about the progress of the verification. The information appears in the log display area at the bottom of the window. The full log displays by default. It display messages, errors, and statistics for all phases of the verification. You can search the full log

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

7 Click the **Compile Log** button to display compile phase messages and errors. You can search the log by entering search terms in the **Search in the log** box and clicking the left arrows to search backward or the right arrows to search forward.

8 Click the **Stats** button to display statistics, such as analysis options, stubbed functions, and the verification checks performed.

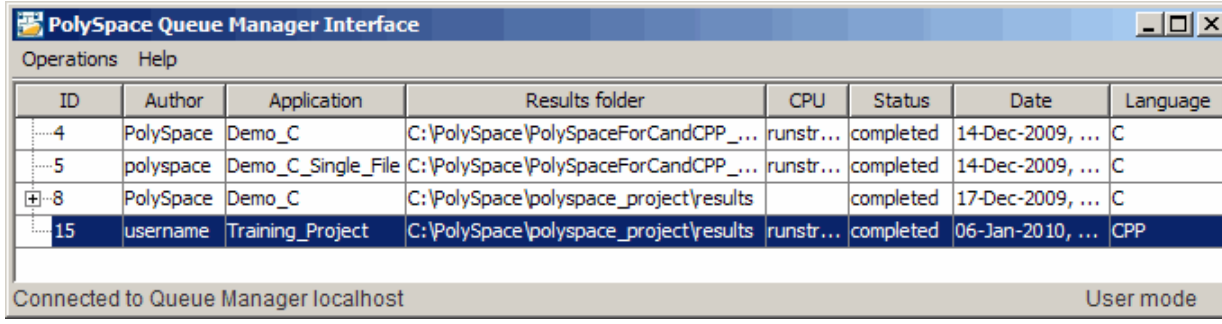
9 Click the refresh button

to update the stats log display as the verification progresses.

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

11 Wait for the verification to complete.

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

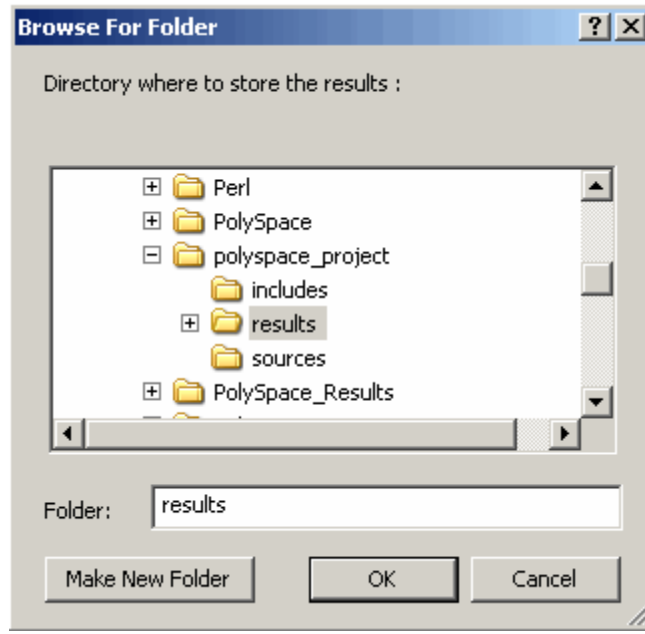


Downloading Results from the Server to the Client

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

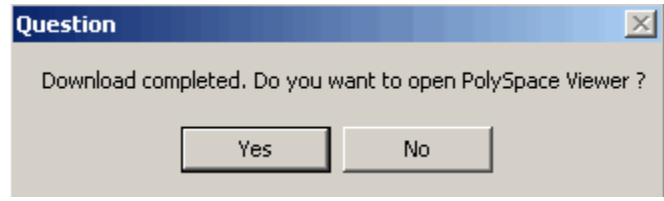
1 In the **PolySpace Queue Manager Interface**, select **Download Results** from the context menu for the verification.

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



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

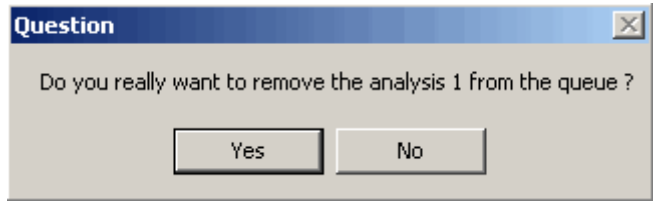
A dialog box appears telling you that the download is complete and asking if you want to open the PolySpace Viewer.



3 Click **No**.

4 Select **Remove From Queue** from the context menu.

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



5 Click **Yes**.

Note

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

---

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

Once the results are on your client, you can review them using the PolySpace Viewer. You review the results from the verification in Chapter 4, “Reviewing Verification Results”.

Troubleshooting a Failed Verification

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

Hardware Does Not Meet Requirements

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

`www.mathworks.com/products/polyspaceclientc/requirements.html`.

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

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

You can:

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

You Did Not Specify the Location of Included Files

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

```
include.h:  No such file or folder
```

For information on how to specify the location of include files, see "Creating a New Project to Verify a Class in the Training C++ File" on page 2-9.

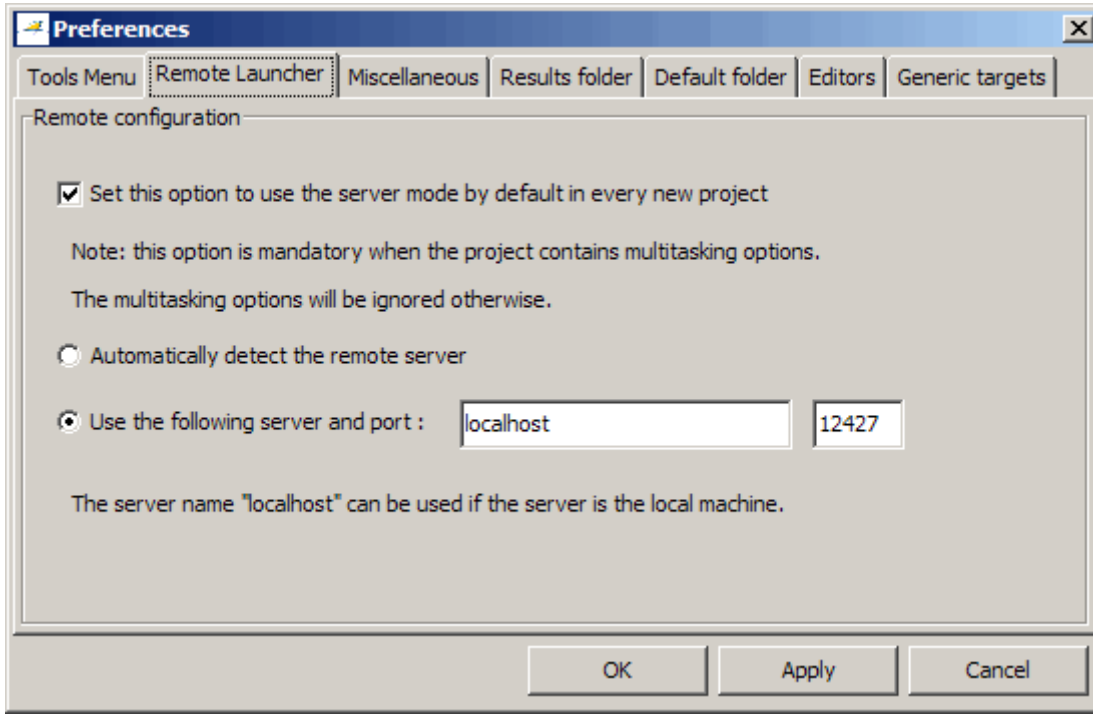
PolySpace Software Cannot Find the Server

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

```
Error:  Unknown host :
```

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

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



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

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

In this section...

Overview of PolySpace In One Click

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

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

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

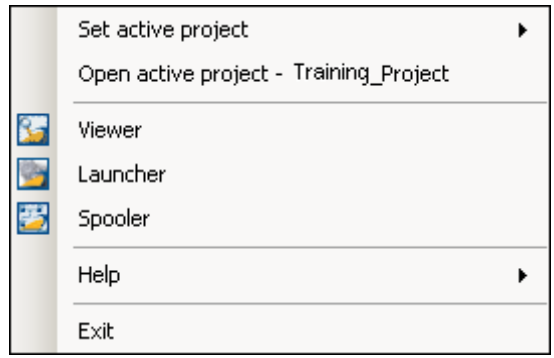
Setting the Active Project

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

To set the active project:

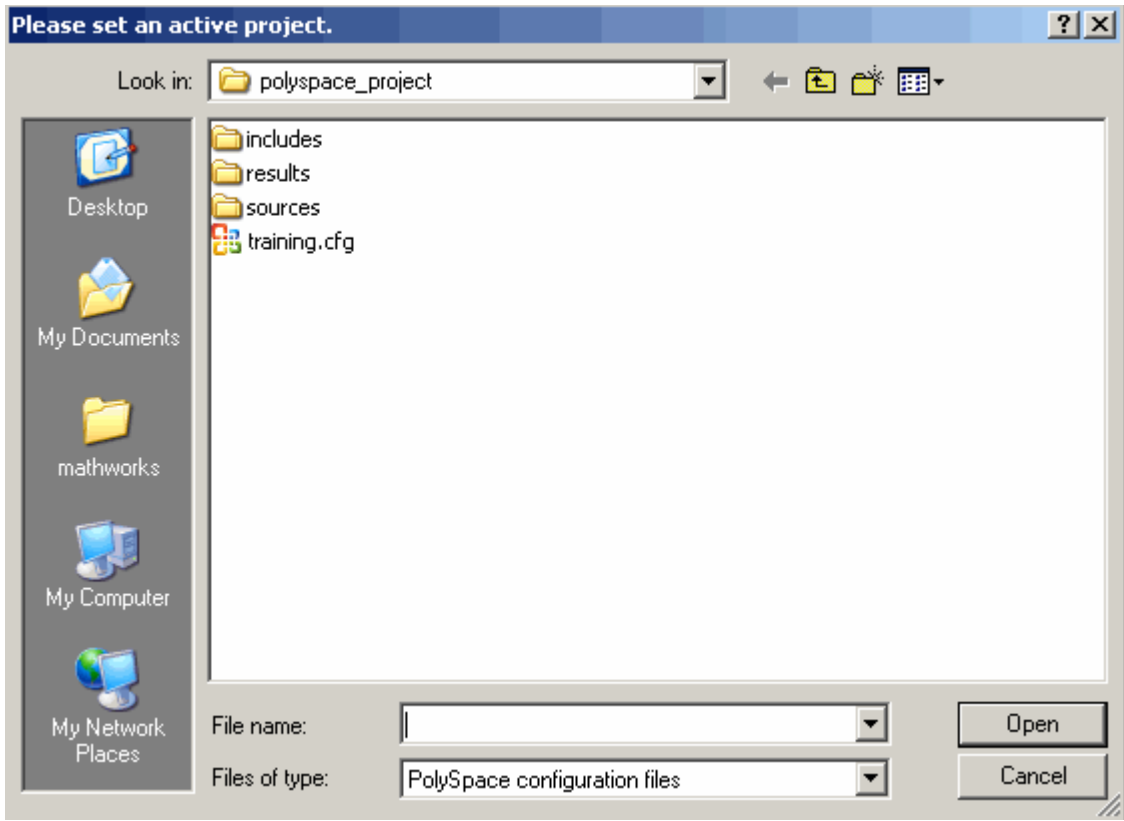
1. Right-click the PolySpace In One Click icon in the taskbar area of your Windows desktop:

The context menu appears.



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

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



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

4 Select `training.cfg`.

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

Sending the Files to PolySpace Software

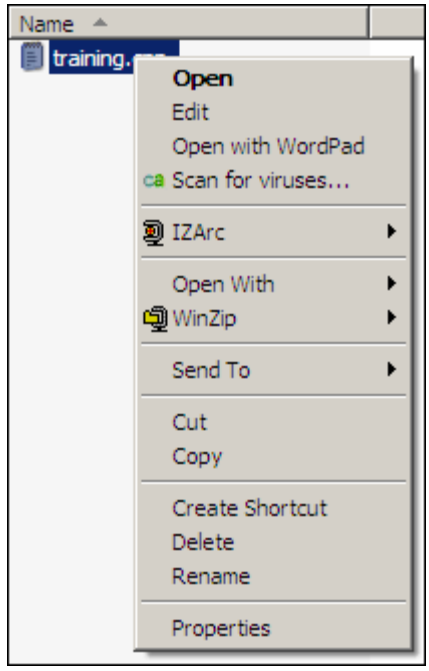
You can send several files to PolySpace software for verification. For this tutorial, you send one file, `training.cpp`.

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

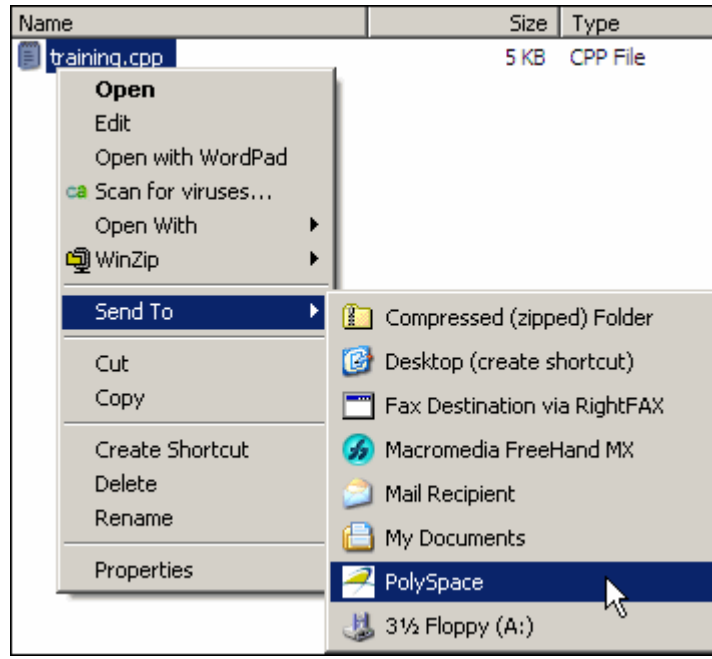
1 Navigate to the folder `polyspace_project\sources`.

2 Right-click the file `training.cpp`.

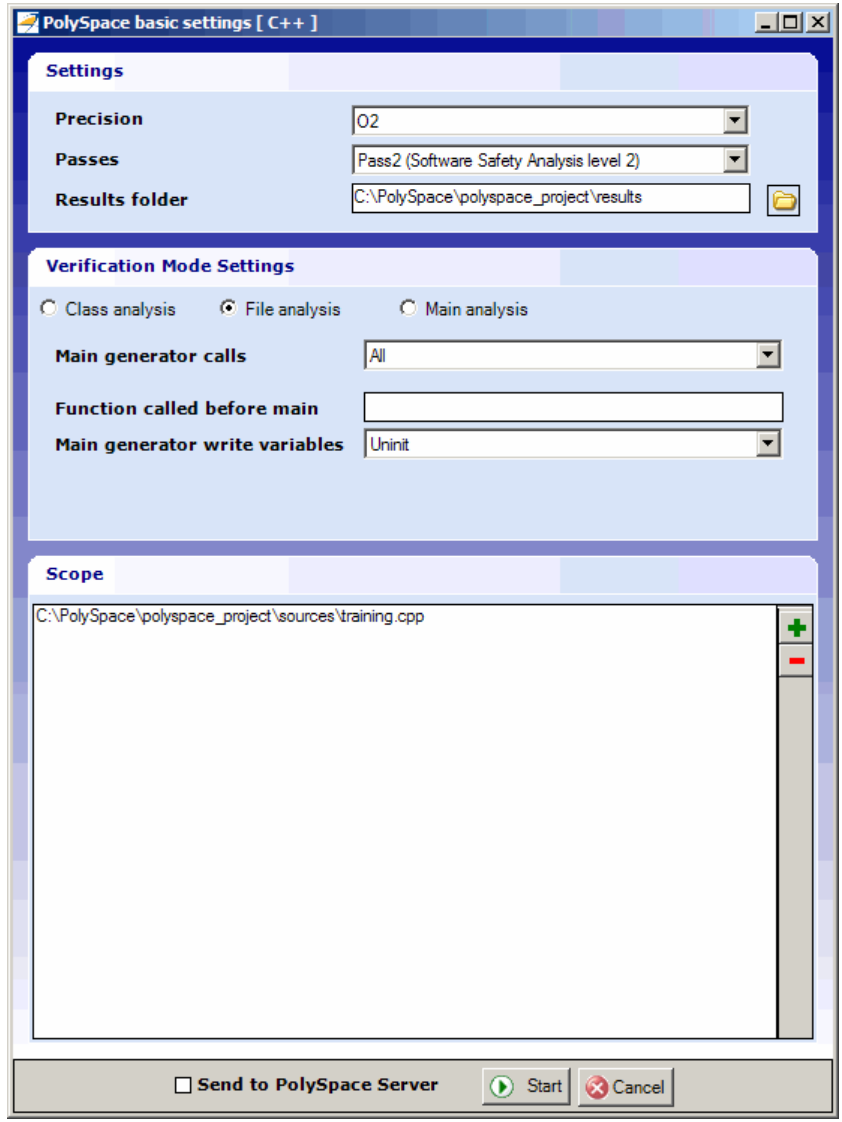
The context menu appears.



3 Select **Send To > PolySpace**.



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



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

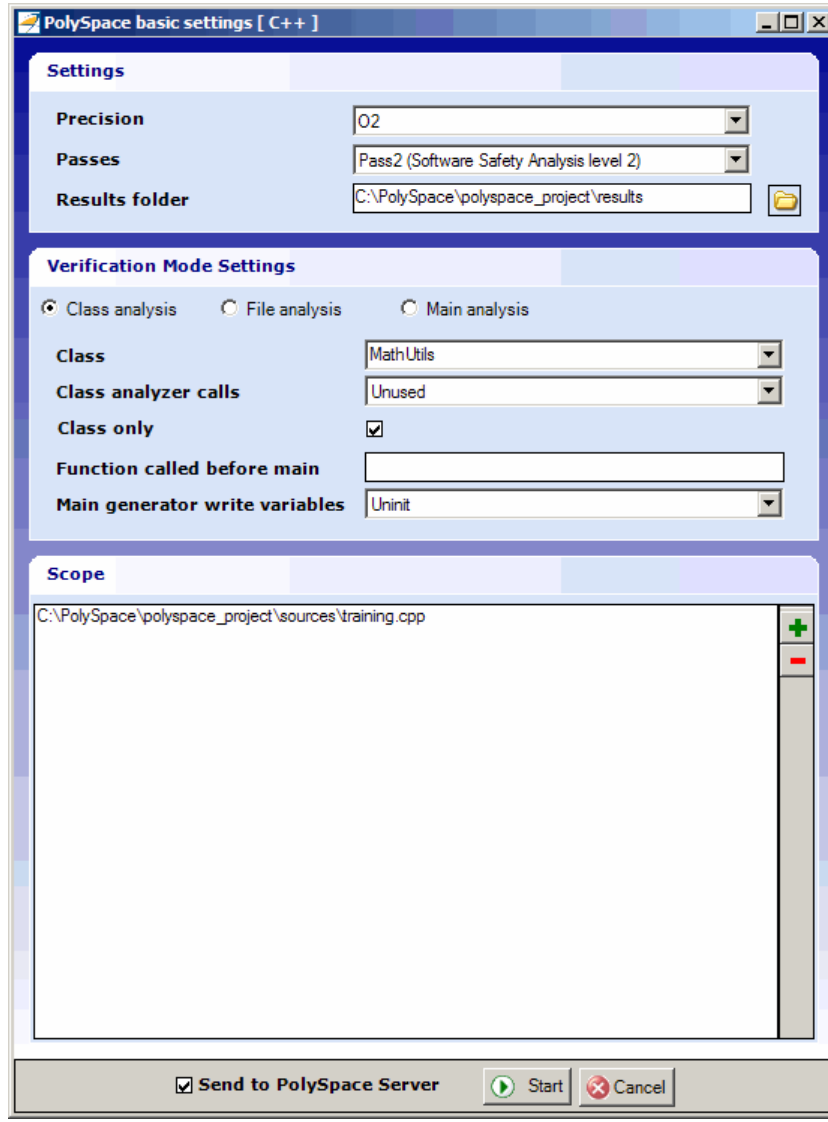
5 You will see that there are three different tabs in the **Parameters** section to assist you in setting up the type of verification you want to run. In this

tutorial, you are verifying a single class, so you want to use the **Class analysis** tab to set up the analysis parameters. Under the **Class analysis** tab, type `MathUtils` in the box labeled **Class**. You will see that 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.

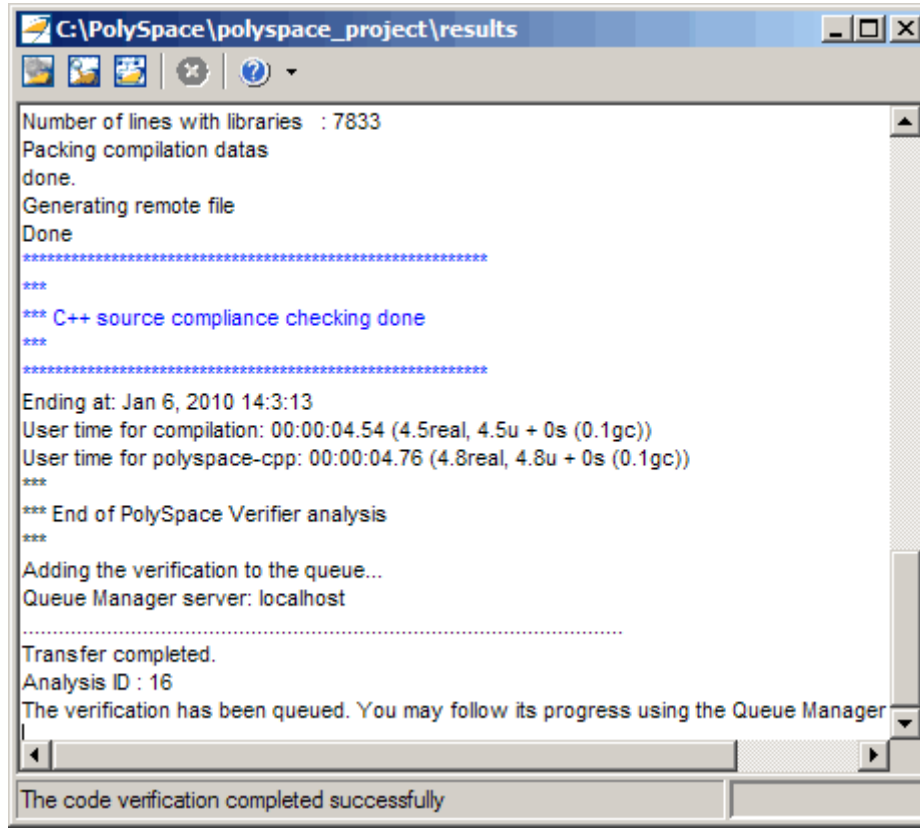
7 Leave the default values for the other parameters.

The **PolySpace basic settings window** should now look like this.



Click **Start**.

The verification log appears.



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

- You see the message:

  ```
  End of PolySpace Verifier analysis
  ```

- A message in the log area tells you that the verification was transferred to the server and gives you the identification number (Analysis ID) for the verification. For this verification, the identification number is 1.
- Monitor the verification using the Spooler. For information on using the Spooler to monitor a verification on a server, see “Monitoring the Progress of the Verification” on page 3-7.

- When the verification completes, download the results to `polyspace_project\results`. For information on downloading results from a server to a client, see "Downloading Results from the Server to the Client" on page 3-10

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

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

In this section...

Starting the Verification

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

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

To start a verification that runs on a client:

1. Open the Launcher if it is not already open.
2. Open the project file `training.cfg` if it is not already open.

   For information about opening a project file, see "Opening the Project" on page 3-4.
3. Make sure that the **Send to PolySpace Server** check box is clear.
4. If you see a warning that multitasking is not available when you run a verification on the client, click **OK** to continue and close the message box. This warning only appears when you clear the **Send to PolySpace Server** check box.
5. Click the **Start** button.

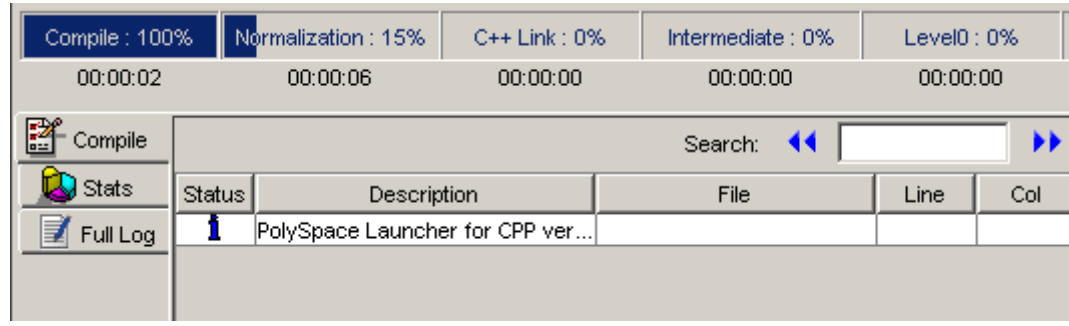
6 If you see a caution that PolySpace software will remove existing results from the results folder, click **Yes** to continue and close the message dialog box.

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

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

Monitoring the Progress of the Verification

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



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

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

1 The compile log displays by default.

This log displays compile phase messages and errors. You can search the log by entering search terms in the **Search in the log** box and clicking the left arrows to search backward or the right arrows to search forward.

2. Click the **Stats** button to display statistics, such as analysis options, stubbed functions, and the verification checks performed.

3. Click the refresh button

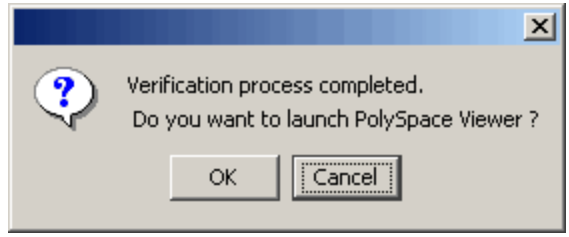
to update the display as the verification progresses.

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

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

Completing the Verification and Stopping the Launcher

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



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

- Click **Cancel**.

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

To close the Launcher:

- Select **File > Quit**.

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

Stopping the Verification Before It Completes

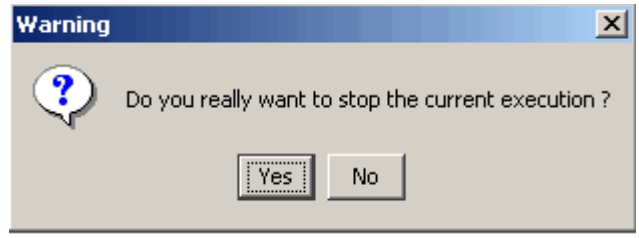
You can stop the verification before it completes. If you stop the verification, results will be incomplete, and if you start another verification, the verification starts over from the beginning.

To stop a verification:

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

   A warning dialog box appears.

2. Click **Yes**.

   The verification stops and the message `Verification process stopped` appears.

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

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

4

Reviewing Verification Results

About This Tutorial

In this section...

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.

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

1. How to use the Viewer, including how to:
   - Open the Viewer and open verification results.
   - Select the Viewer mode.
   - Explore results in expert mode.
   - Explore results in assistant mode.
   - Generate reports.
2. How to interpret the color-coding that PolySpace software uses to identify the severity of an error.
3. How to find the location of an error in the source code.

Before You Start

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

`polyspace_project\results\RTE_px_O2_Training_Project_LAST_RESULTS.rte`.

Opening the Viewer and the Verification Results

In this section...

Opening the Viewer

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

To open the Viewer:

- Double-click the PolySpace Viewer icon:

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

Selecting the Viewer Mode

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

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

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

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

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

Opening the Results

To open the verification results:

1. Select **File > Open**.
2. In the **Please select a file dialog box**, navigate to `polyspace_project\results` and select the file `RTE_px_02_Training_Project_LAST_RESULTS.rte`.
3. Click the **Open** button.

   The results appear in the Viewer window.

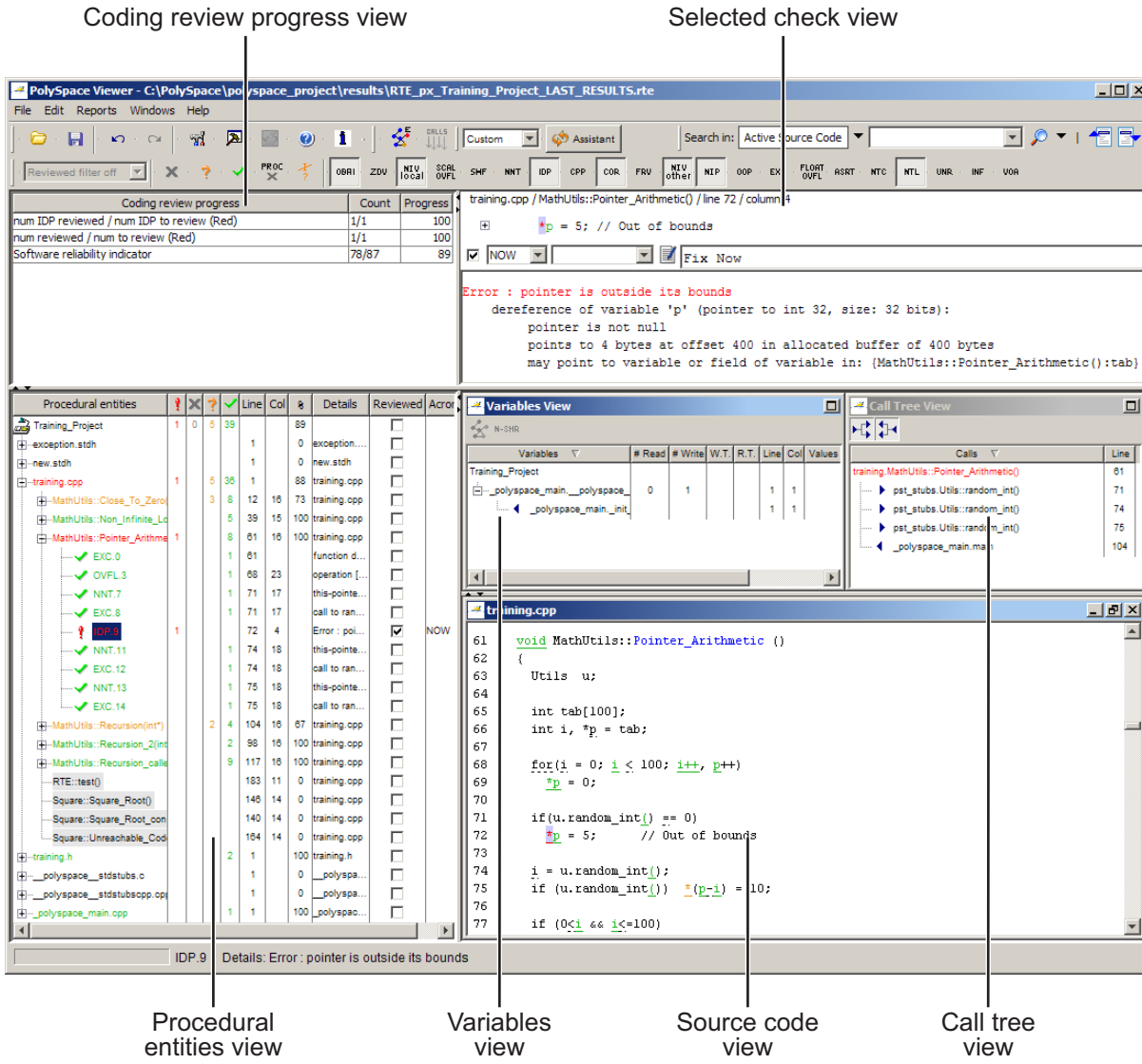
Note The file `RTE_px_02_Training_Project_LAST_RESULTS.rte` represents the verification with the highest level of precision. The lower level results files that you see in the `polyspace_project\results` folder represent lower precision verifications.

Exploring the Viewer Window

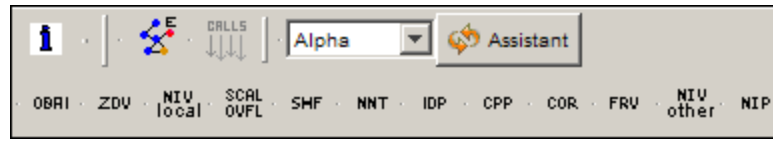
In this section...

Overview

The PolySpace Viewer window looks like this.



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



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

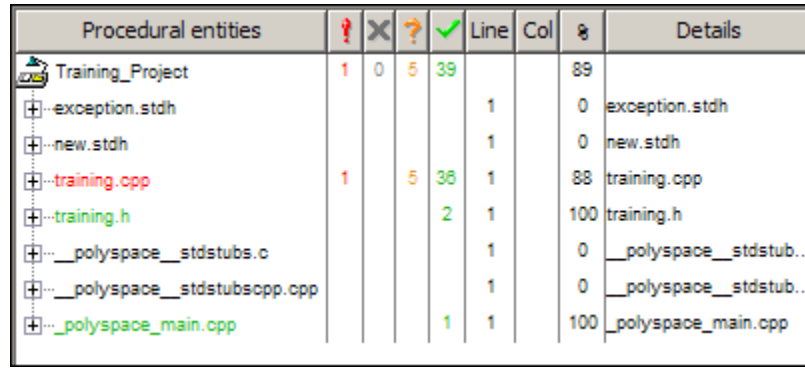
This view...	Displays...
Procedural entities view (lower left)	List of the diagnostics (checks) for each file and function in the project
Source code view (lower right)	Source code for a selected check in the procedural entities view
Coding review progress view (upper left)	Statistics about the review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

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

Reviewing the Procedural Entities View

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

Error) view. When you first open the results file from the verification of `training.cpp`, the procedural entities view looks like this.



Procedural entities	!	✗	?	✓	Line	Col	%	Details
Training_Project	1	0	5	39			89	
exception.stdh					1		0	exception.stdh
new.stdh					1		0	new.stdh
training.cpp	1		5	38	1		88	training.cpp
training.h				2	1		100	training.h
__polyspace__stdstubs.c					1		0	__polyspace__stdstub...
__polyspace__stdstubscpp.cpp					1		0	__polyspace__stdstub...
_polyspace_main.cpp				1	1		100	_polyspace_main.cpp

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

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

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

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

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

Reviewing Results in Expert Mode

In this section...

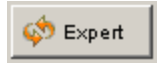
What Is Expert Mode?

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

Switching to Expert Mode

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

- Click the Viewer mode button:

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

Reviewing Checks in Expert Mode

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

- "Selecting a Check to Review" on page 4-11
- "Displaying the Calling Sequence" on page 4-12
- "Tracking Review Progress" on page 4-13

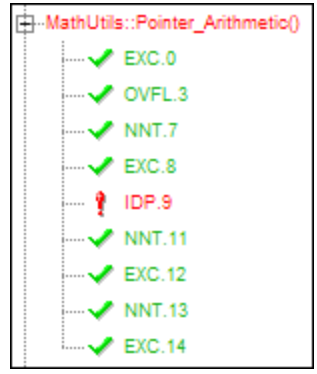
- "Tracking Reviewed Checks in Procedural Entities View" on page 4-15

Selecting a Check to Review

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

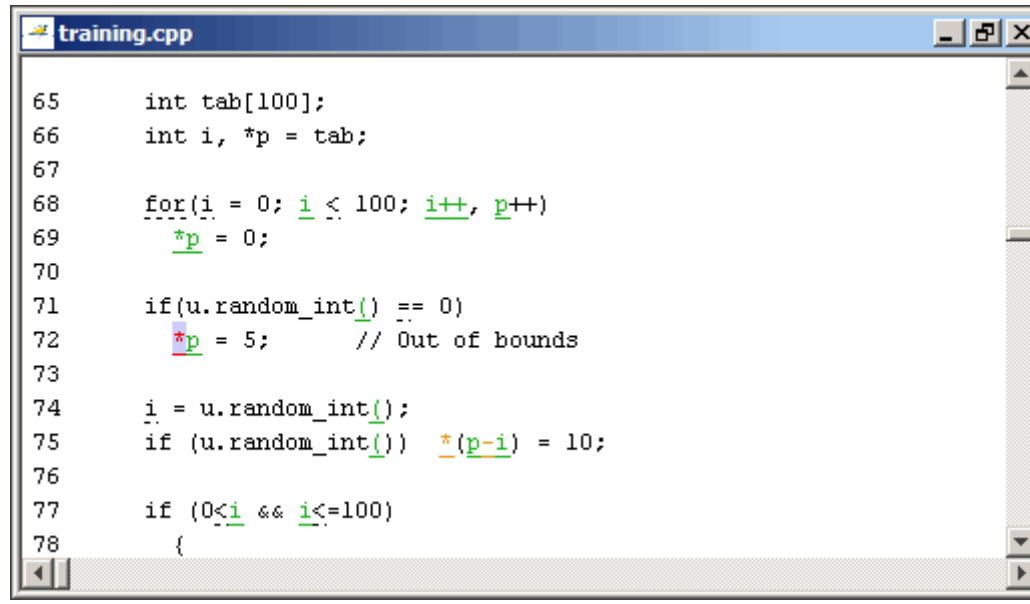
1. In the procedural entities section of the window, expand `training.cpp`.
2. Expand the red procedure `MathUtils::Pointer_Arithmetic()`.

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



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

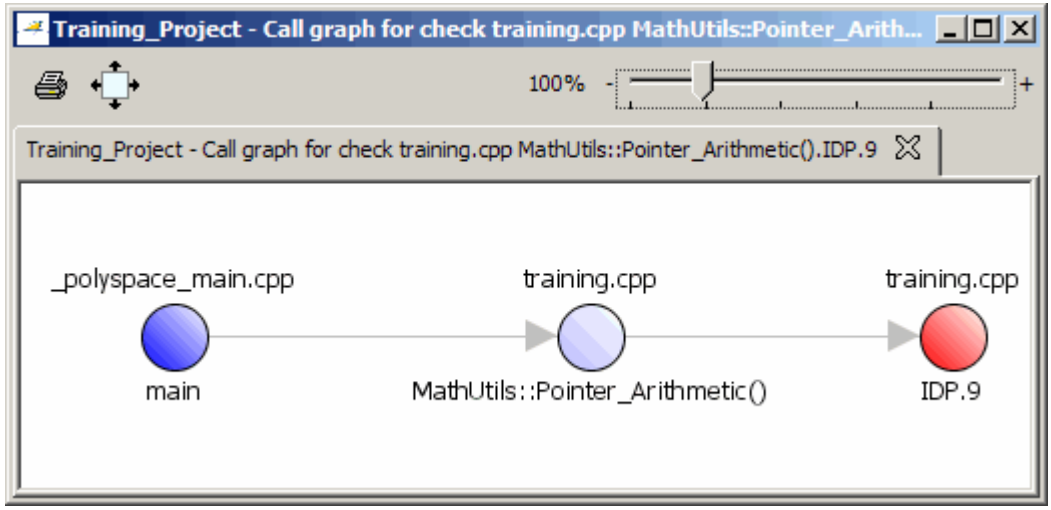
You can display the calling sequence that leads to the code associated with a check. To see the calling sequence for the red `IDP.9` check in `MathUtils::Pointer_Arithmetic()`:

1 Expand `MathUtils::Pointer_Arithmetic()`.

2 Click the red `IDP.9`.

3 Click the call graph button in the 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`.

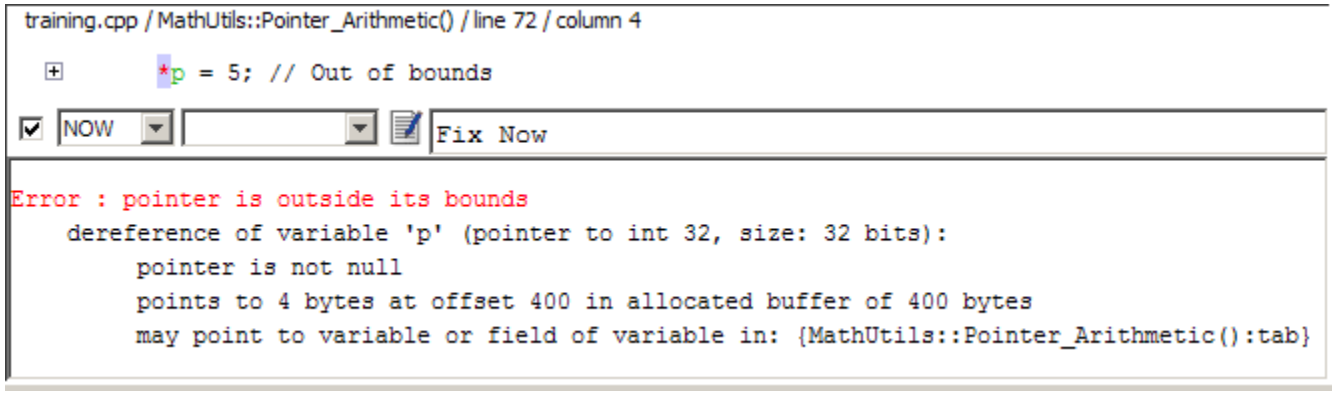
   A table with statistics about the review progress for that category and severity of error appear in the upper-left part of the window.

Coding review progress	Count	Progress
num IDP reviewed / num IDP to review (Red)	0/1	0
num reviewed / num to review (Red)	0/1	0
Software reliability indicator	78/87	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 third row displays the ratio of the number of green checks to the total number of checks, providing an indicator of the reliability of the software.

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



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

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

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

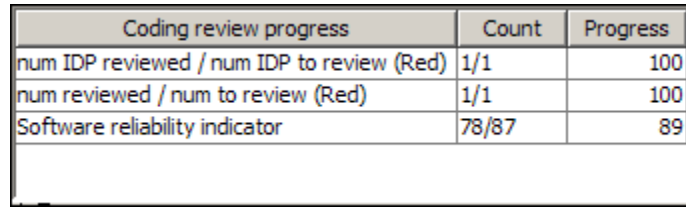
Note You can also define your own acronyms. See "Defining Custom Acronyms ".

---

4 In the comment box, enter additional information about the check.

5 Select the check box to indicate that you have reviewed this check.

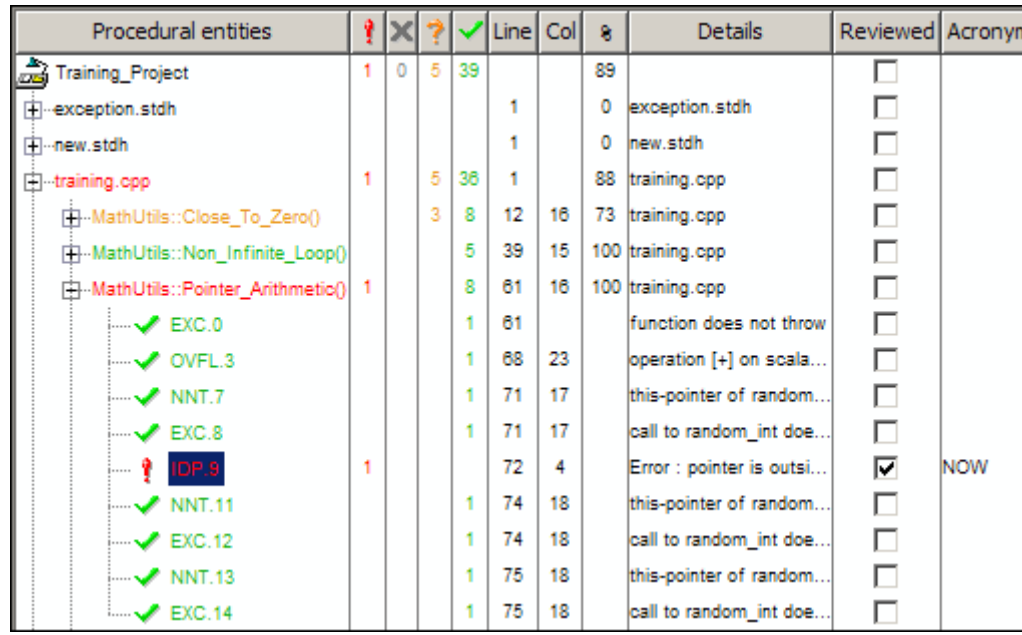
The **Coding review progress** part of the window updates the ratios of errors reviewed to total errors.



Coding review progress	Count	Progress
num IDP reviewed / num IDP to review (Red)	1/1	100
num reviewed / num to review (Red)	1/1	100
Software reliability indicator	78/87	89

Tracking Reviewed Checks in Procedural Entities View

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



Procedural entities	!	✗	?	✓	Line	Col	%	Details	Reviewed	Acronym
Training_Project	1	0	5	39			89		☐	
exception.stdh					1		0	exception.stdh	☐	
new.stdh					1		0	new.stdh	☐	
training.cpp	1		5	36	1		88	training.cpp	☐	
MathUtils::Close_To_Zero()			3	8	12	16	73	training.cpp	☐	
MathUtils::Non_Infinite_Loop()				5	39	15	100	training.cpp	☐	
MathUtils::Pointer_Arithmetic()	1			8	61	16	100	training.cpp	☐	
EXC.0				1	61			function does not throw	☐	
OVFL.3				1	68	23		operation [+] on scala...	☐	
NNT.7				1	71	17		this-pointer of random...	☐	
EXC.8				1	71	17		call to random_int doe...	☐	
IDP.9	1				72	4		Error : pointer is outsi...	☑	NOW
NNT.11				1	74	18		this-pointer of random...	☐	
EXC.12				1	74	18		call to random_int doe...	☐	
NNT.13				1	75	18		this-pointer of random...	☐	
EXC.14				1	75	18		call to random_int doe...	☐	

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

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

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

Reviewing Additional Examples of Checks

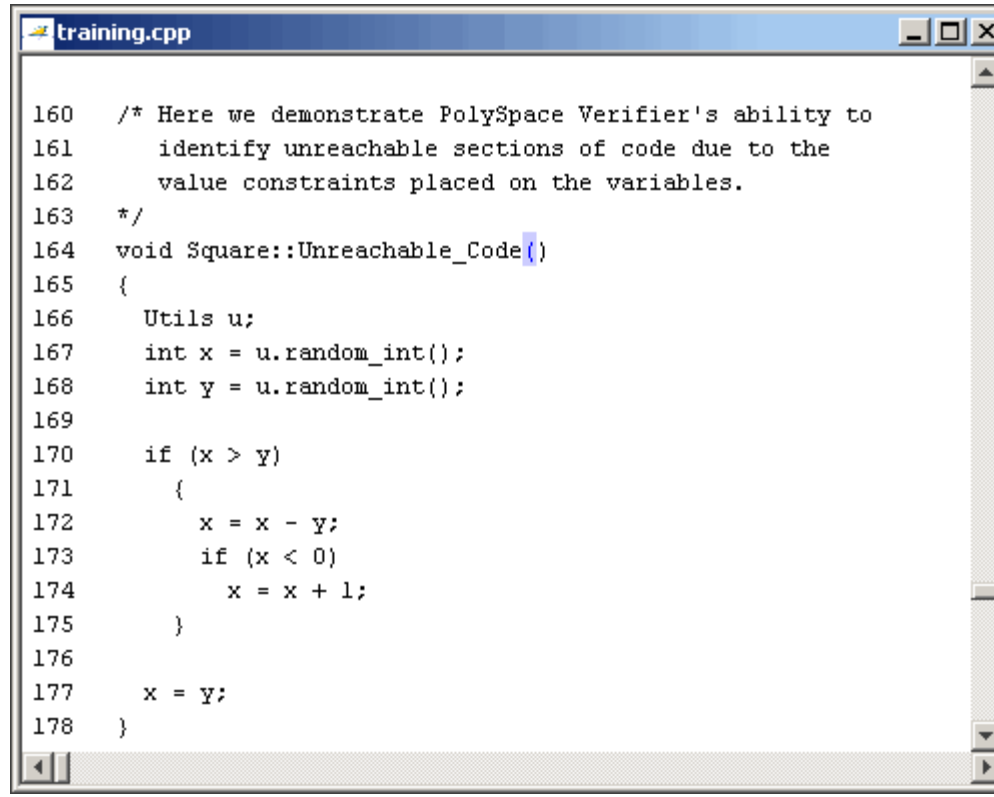
In this part of the tutorial, you learn about other types and categories of errors by reviewing the following examples in `training.cpp`:

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.



2 Examine the source code.

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

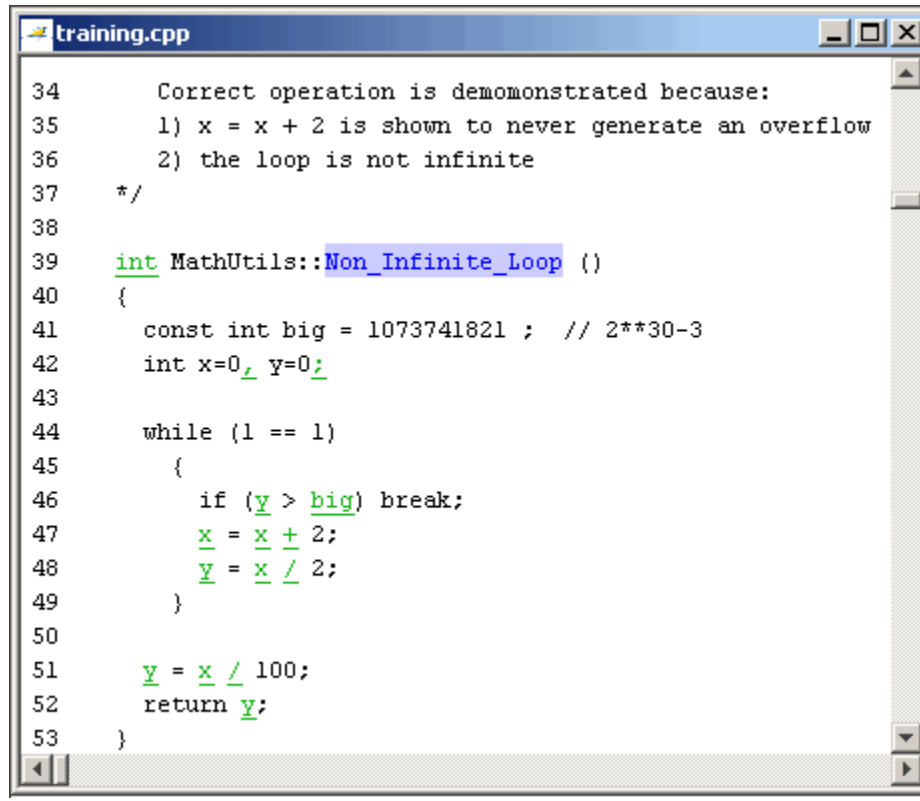
Note that in the **Procedural Entities** view all public and protected member functions for the classes `RTE` and `Square` are marked as unreachable code. This is because the analysis results are from the single class verification of `MathUtils` which does not depend on any other classes.

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.



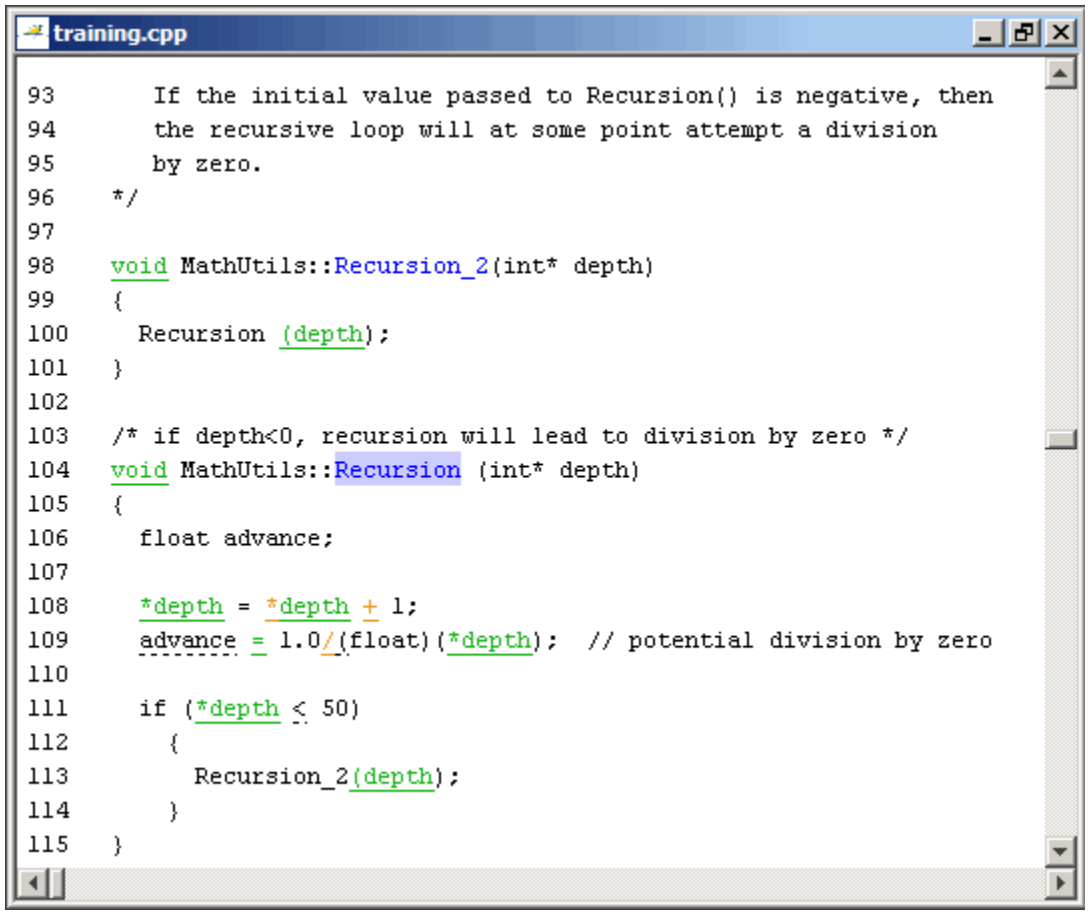
2 Examine the source code. The variable `x` never overflows because the `while` loop at line `44` terminates before x can overflow.

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.



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 the Types of Checks That You See

You can filter the checks that you see in the Viewer so that you can focus on certain types of checks. PolySpace software provides three predefined

composite filters, a custom composite filter, and several individual filters. You learn about filters in the following sections:

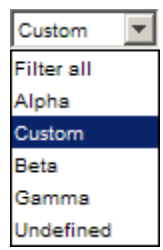
- "Using Composite Filters" on page 4-21
- "Using the Custom Filter" on page 4-22
- "Using Individual Filters" on page 4-25

Using Composite Filters

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

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

The default filter is `Custom`. You learn more about the `Custom` filter in "Using the Custom Filter" on page 4-22. You can select a composite filter from the filter menu.

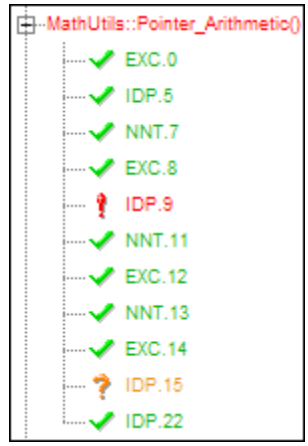


To learn how the composite filters affect the display of checks:

1 Expand the function `MathUtils::Pointer_Arithmetic()` in **Procedural Entities**. Select `Alpha` from the filter menu to display all the checks for `MathUtils::Pointer_Arithmetic()`.

`MathUtils::Pointer_Arithmetic()` has twenty-four checks: twenty-two green, one red, and one orange.

2 Select `Beta` from the filter menu to hide the NIV local, SCAL OVFL, NIV other, NIP, and FLOAT OVFL checks.



Now, only eleven checks are visible: four EXC, four IDP, and three NNT.

3 Select `Alpha` to display all checks again.

4 Select `Gamma` to display only the red and gray checks.

Now, only one check is visible: the red IDP.

Using the Custom Filter

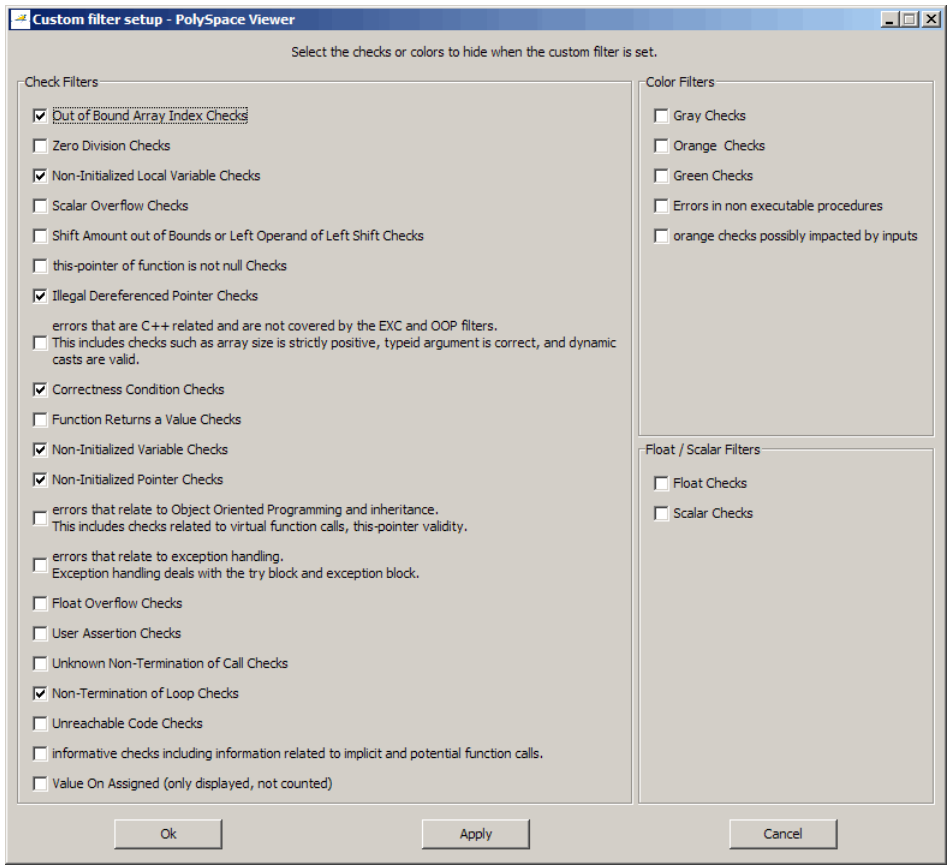
The custom filter is a composite filter that you define. It appears on the composite filter menu as `User def` and is the default composite filter. By

default, the custom filter hides the OBAI, NIV local, IDP, COR, IRV, NIV other, NIP, and NTL checks as shown in the following figure.

To modify the custom filter:

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

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



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

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

4 Select the filters for the checks that you do not want to display.

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

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

Using Individual Filters

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

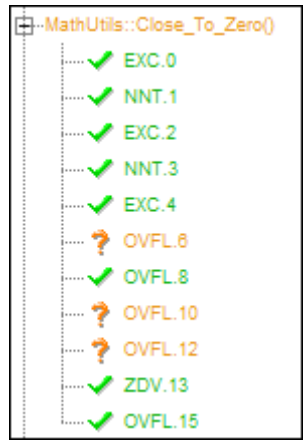
Tip When you mouse over a filter button, a tooltip tells you which filter the button is for and whether the filter is currently enabled or disabled.

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

1. Expand `MathUtils::Close_To_Zero()`.
2. Select `Alpha` from the composite filters menu to display all checks.
3. Click the **NIV local** filter button

   to hide the NIVL checks for `MathUtils::Close_To_Zero()`.



4 Click the **NIV local** filter button again to display the NIVL checks.

5 Now click the green checks filter button

to hide the green 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()`.

Reviewing Results in Assistant Mode

In this section...

What Is Assistant Mode?

In assistant mode, PolySpace software chooses the checks for you to review and the order in which you review them. PolySpace software presents checks to you in this order:

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

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

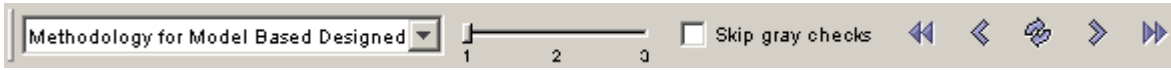
Switching to Assistant Mode

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

- Click the Viewer's switch mode button 

.

  The Viewer window toolbar displays controls specific to assistant mode.



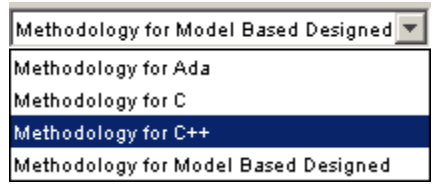
The controls for assistant mode include:

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

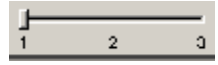
Selecting the Methodology and Criterion Level

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

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



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



Exploring Methodology for C++

In this part of the tutorial, you examine the configuration for **Methodology for C++**. To begin:

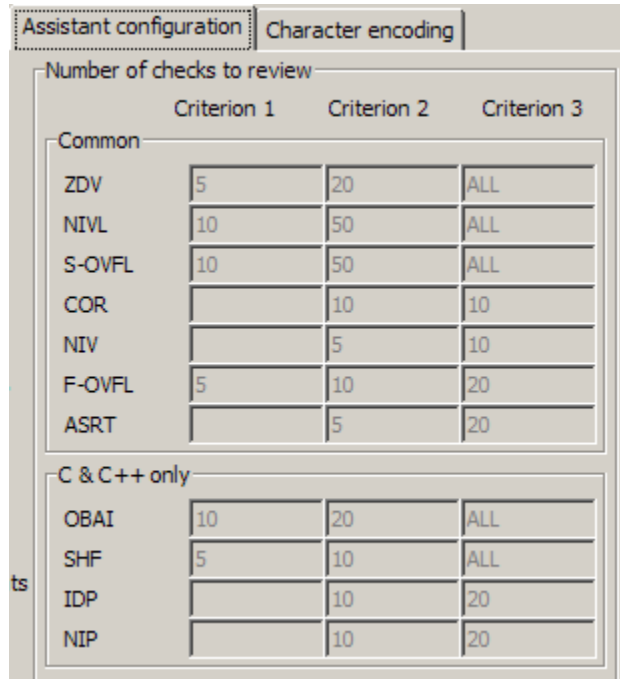
1 Select **Edit > Preferences**.

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

2 Select the **Assistant configuration** tab.

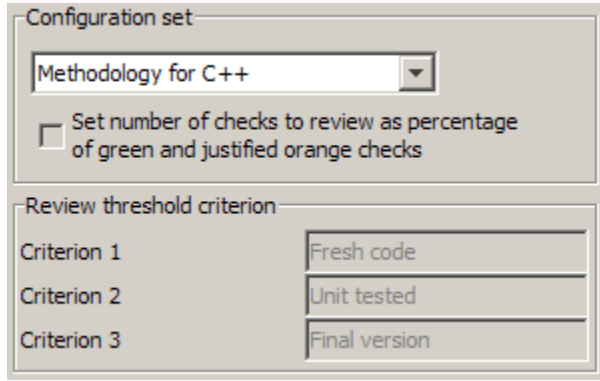
The configuration for Methodology for C++ appears.

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



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

In the lower-left part of the dialog box, the section **Review threshold criterion** contains text that appears in the tooltip for the criterion slider on the Viewer toolbar (only in assistant mode).



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

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

1 All reds

2 All blocks of gray checks (the first check in each unreachable function)

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

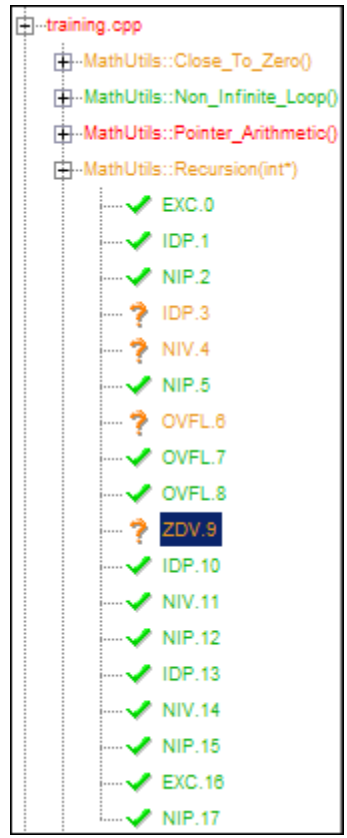
---

3 Orange checks according to the selected methodology and criterion level

Earlier in this tutorial, you selected Methodology for C++, criterion l. 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. In the procedural entities view (lower left), `MathUtils::Recursion(int*)` is expanded and `ZDV.9` is displayed as the current check.

   If the Viewer is displaying the message "No check currently selected" in the upper-right portion of the window, then you will need to click the forward arrow  to go to the first check.

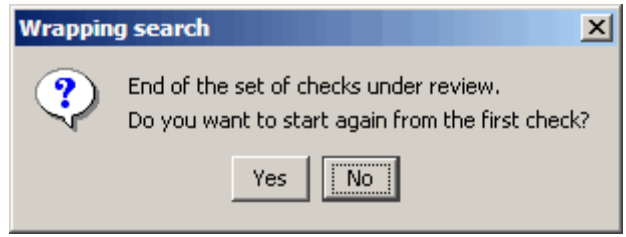


   The source code view (lower right) displays the source for this check and the current check view (upper right) displays information about this check.

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

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

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



3 Click **No**.

Defining a Custom Methodology

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

To define your custom methodology:

1 Select **Edit > Preferences**.

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

2 Select the **Assistant configuration** tab.

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

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

5 Under the **Criterion 1** column, enter the number `1` next to **IDP**. This tells PolySpace software to select up to one orange IDP for review. Poly Space™ 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.

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

Generating Reports of Verification Results

In this section...

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 JSF Coding Rules, as well as PolySpace configuration settings used 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 used for the verification.
- **Developer with Green Checks Report** – Provides the same content as the Developer Report, but also includes a detailed list of green checks.
- **Quality Report** – Provides information useful to quality engineers, including summary results, statistics about the code, graphs showing distributions of checks per file, and PolySpace configuration settings used for the verification.

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

- HTML
- PDF
- RTF
- WORD
- XML

Note WORD format is not available on UNIX platforms, RTF format is used instead.

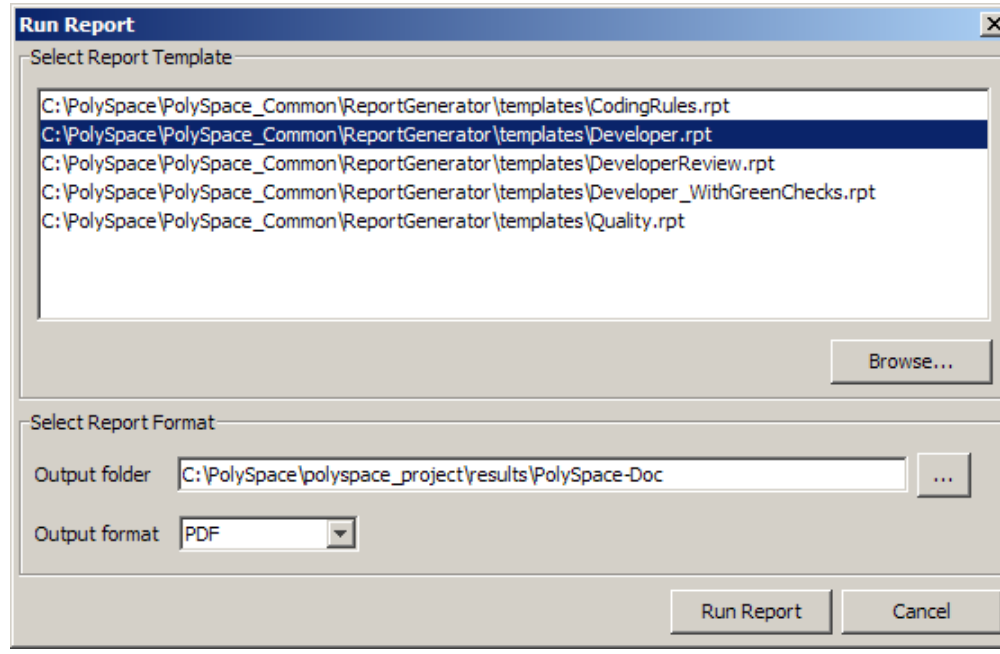
Generating Verification Reports

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

To generate a verification report:

1. Open your verification results in the Viewer.
2. Select **Reports > Run Report**.

   The Run Report dialog box opens.



3 Select the type of report you want to run in the Select Report Template section.

4 Select the Output folder in which to save the report.

5 Select the Output format for the report.

6 Click **Run Report**.

The software creates the specified report.

5

Checking Compliance with Coding Rules

About This Tutorial

In this section...

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. Set an option for checking JSF++ compliance.
2. Select JSF++ rules to check.
3. Run a verification with JSF++ checking.

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 Project File".

Setting Up Coding Rules Checking

In this section...

Opening the Example Project

For this tutorial, you modify the project in `training.cfg` to include JSF++ checking and save the project with a new name. You use the Launcher to modify the project.

To open the Launcher:

- Double-click the Launcher icon.

To open `training.cfg`:

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

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

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

3. Select `training.cfg`.

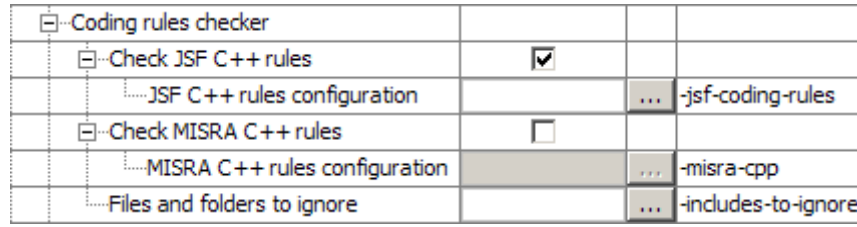
4. Click **Open** to open the file and close the dialog box.

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 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`.



These options allow you to specify which rules to check and any files to exclude from the checker.

2 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 5-4
- "Setting All the Rules to Off" on page 5-6
- "Selecting the Rules to Check " on page 5-6

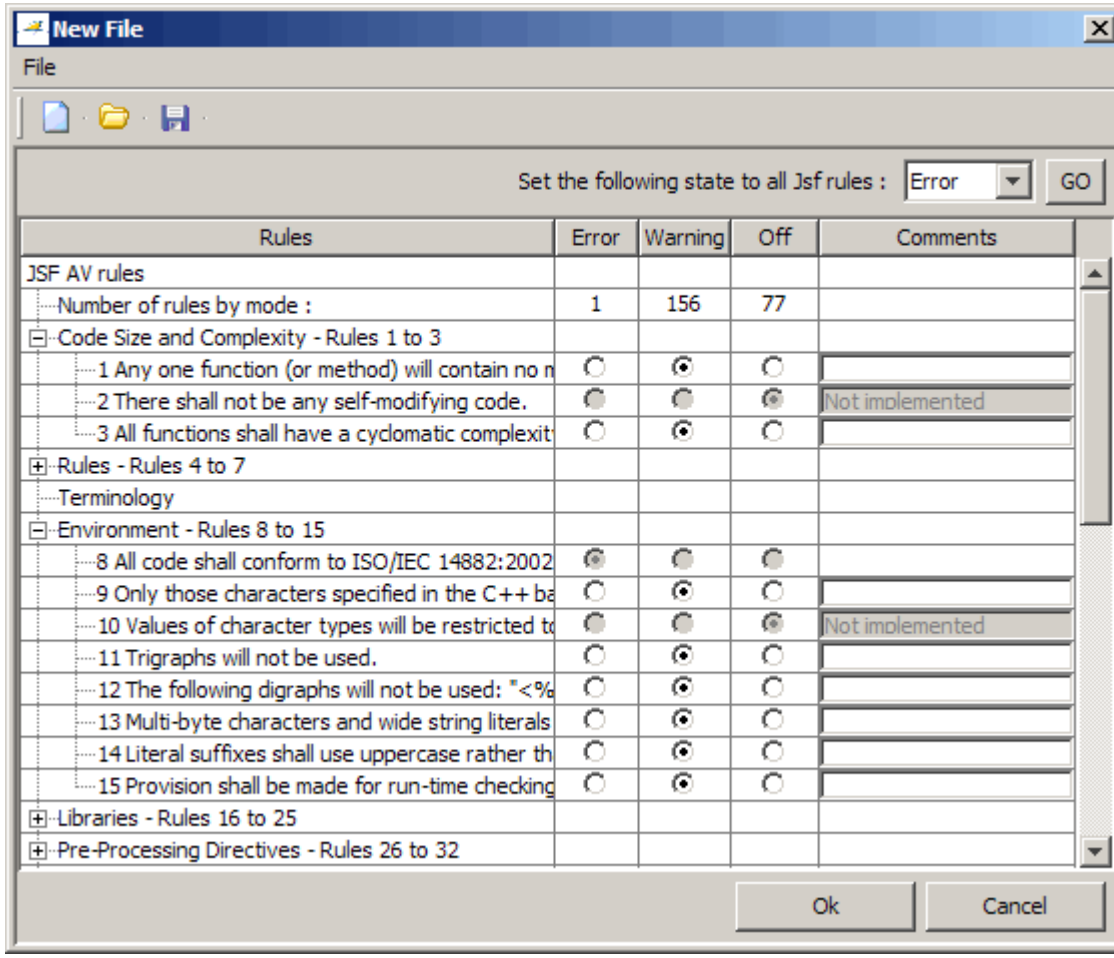
Opening a New Rules File

To open 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.

The New File window opens, allowing you to create a new JSF++ rules file, or open an existing file.



For each JSF++ rule, you specify one of these states:

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

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

Setting All the Rules to Off

Because this tutorial checks only a few rules, first set the state of all rules to `Off`. Later, you select the rules to check.

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

1. From the **Set the following state to all 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.

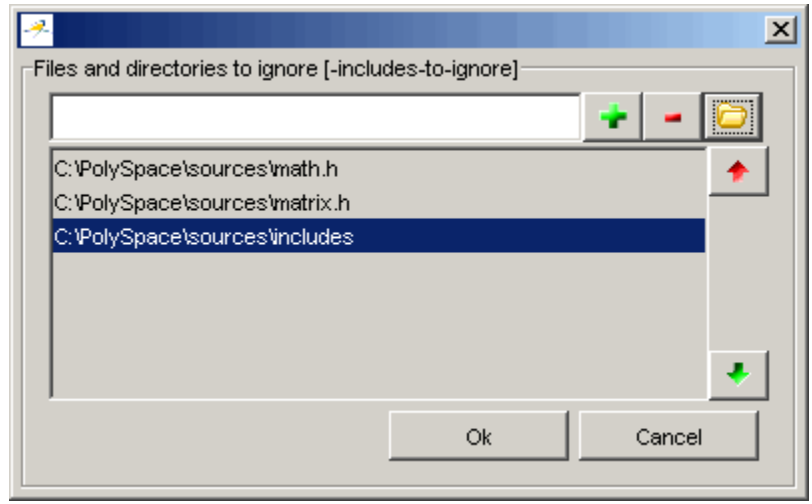
   The **Save as** dialog box opens.
6. In **File**, enter `jsf.txt`
7. Click **OK** to save the file and close the dialog box.

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.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 Launcher. Configuring text and XML editors allows you to view source files and JSF reports directly from the JSF log in the Launcher.

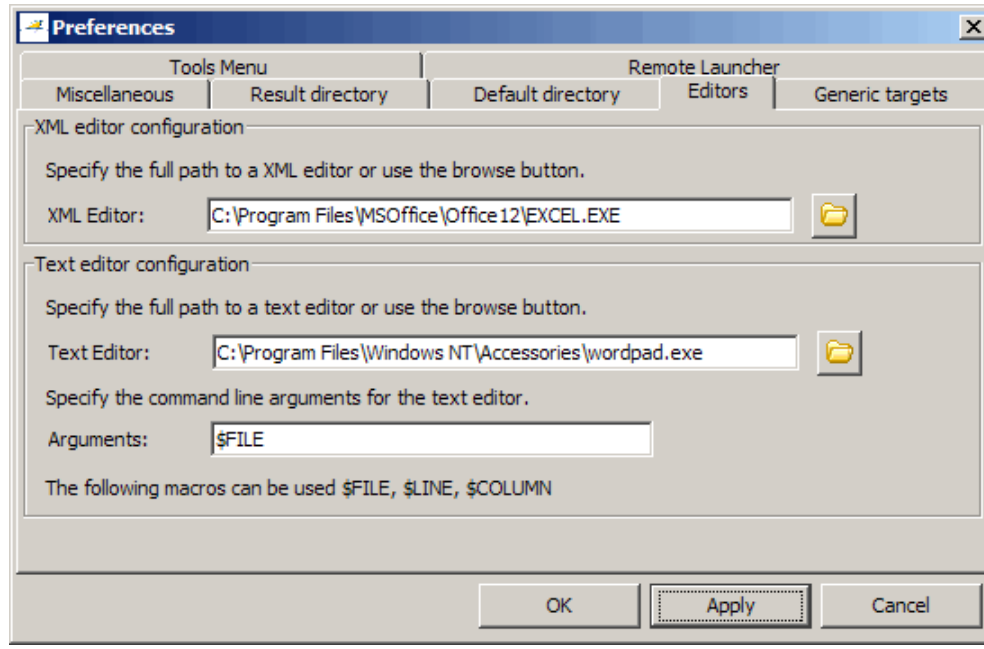
To configure your text and .XML editors:

1 Select **Edit > Preferences**.

The Preferences dialog box opens.

2 Select the **Editors** tab.

The Editors tab opens.



3 Specify an XML editor to use to view 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 with a New Name

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

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

Running a Verification with Coding Rules Checking

In this section...

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.

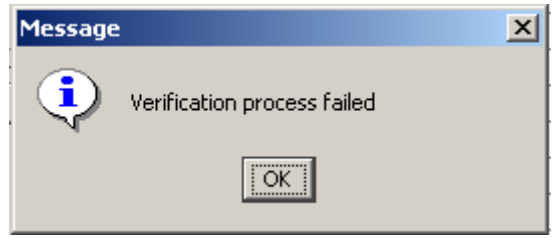
The verification stops if there is a violation of a rule with state `Error`.

To start the verification:

1. Click the **Start** button 

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

   The verification fails because of JSF++ violations. A message dialog box appears.



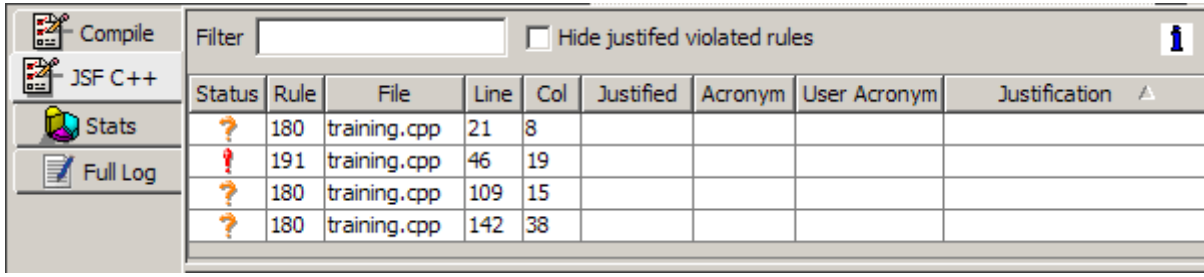
3 Click **OK**.

Examining the JSF Log

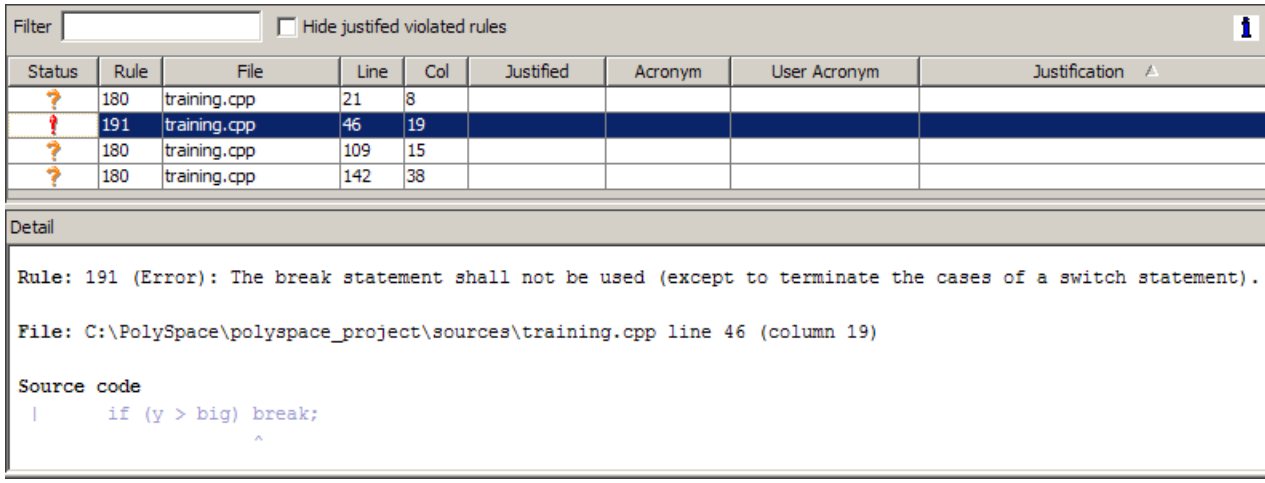
To examine the JSF++ violations:

1 Click the **JSF C++** button in the log area of the Launcher window.

A list of JSF++ violations appear in the log part of the window.

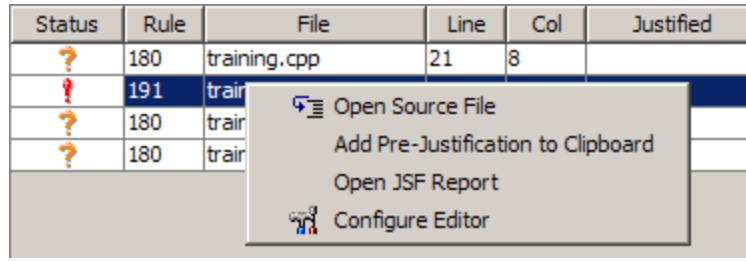


2 Click on any of the violations to see a description of the violated rule, the full path of the file in which the violation was found, and the source code containing the violation.



The log reports a violation of rule 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 5-8.

---

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.

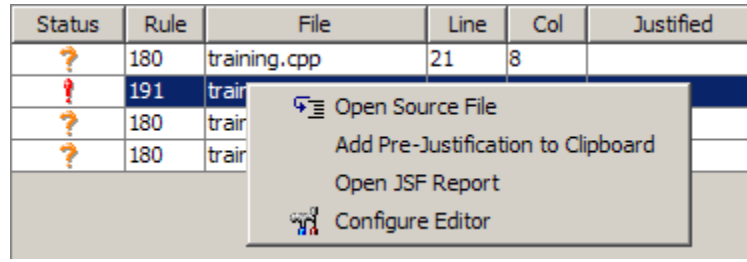
Note You must configure an XML editor before you can open a JSF report. See "Configuring Text and XML Editors" on page 5-8..

To view the JSF report:

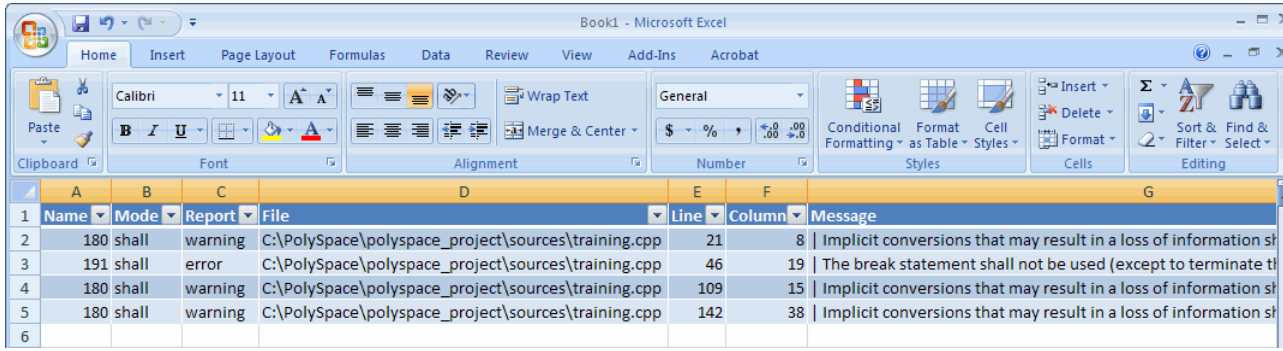
1 Click the **JSF** button in the log area of the Launcher window.

A list of JSF++ violations appears in the log part of the window.

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



The report opens in your XML editor.



6

Using a PolySpace Project Model File

About This Tutorial

In this section...

Overview

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

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

Before You Start

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

Creating a New PolySpace Project Model File

In this section...

What Is a PolySpace Project Model File?

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

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

Creating the PolySpace Project Model File

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

Opening a New Project

To open a new project:

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

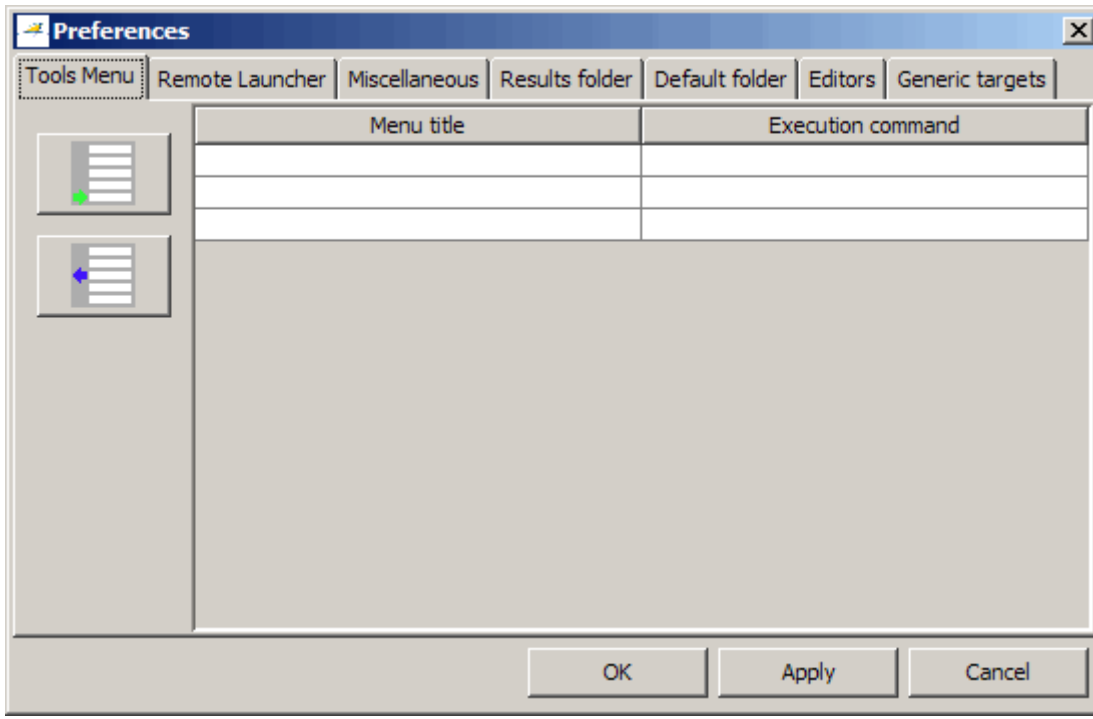
Examining the Preferences Before Adding the Generic Target

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

To look at the generic targets in the preferences:

1. Select **Edit > Preferences**.

   The **Preferences** dialog box appears.



2 Select the **Generic targets** tab.

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

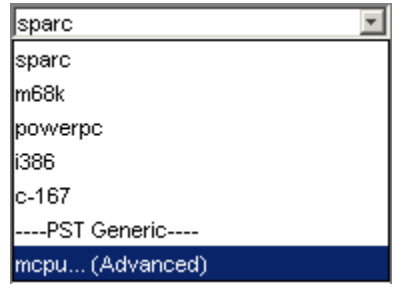
3 Click **Cancel** to close the dialog box.

Defining the Generic Target

To define a generic target:

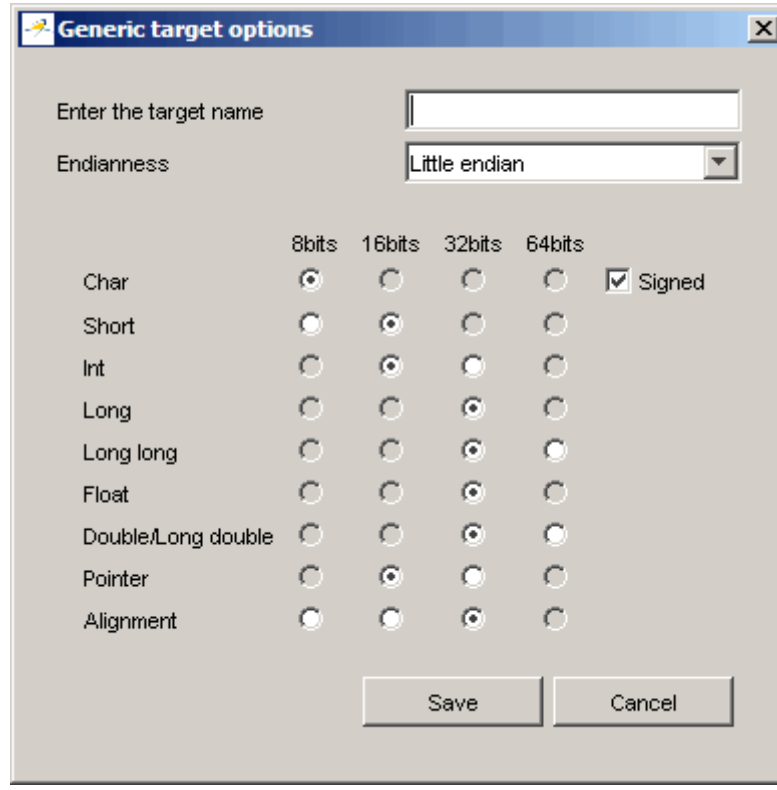
1 In **Analysis options**, expand **Target/Compilation**.

2 Click the down arrow to open the **Target processor type** menu.



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

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



4 In **Enter the target name**, type `target1`.

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

Examining the Preferences After Adding the Generic Target

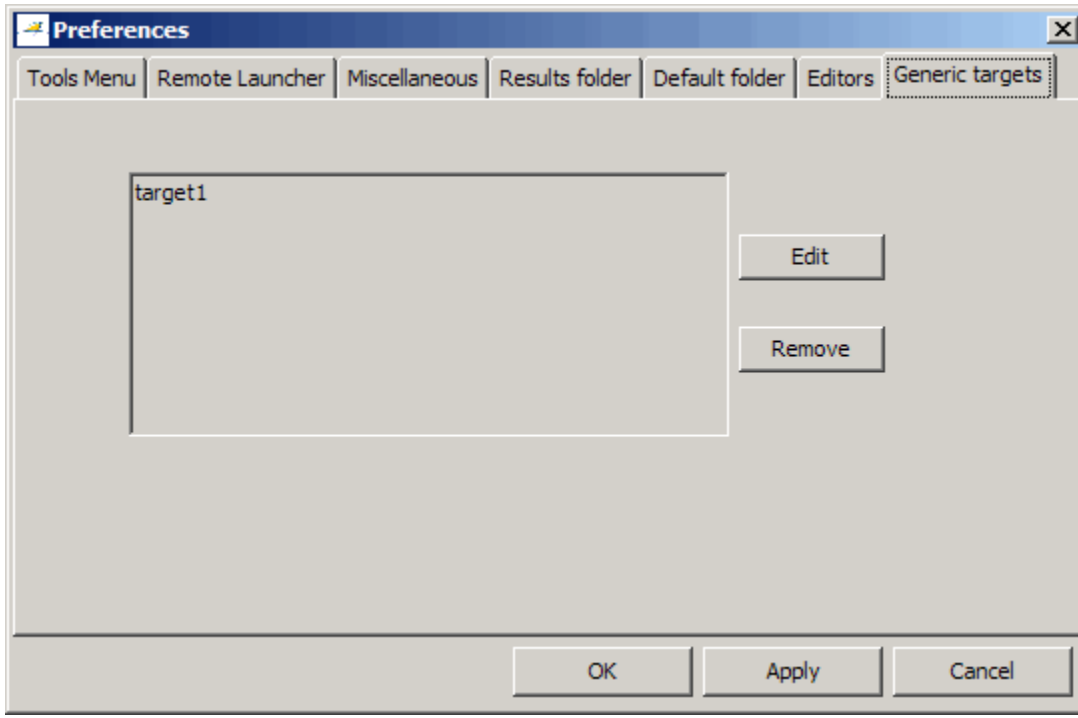
Now when you look at the generic targets in the preferences, you should see the generic target that you added. To look at the generic targets list in the preferences:

1 Select **Edit > Preferences**.

The **Preferences** dialog box appears.

2 Select the **Generic targets** tab.

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



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

Saving the PolySpace Project Model File

To save the PolySpace project model file:

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

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

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

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

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

Warning **The generic target that you defined in this tutorial remains in your preferences until you delete it. Be sure to complete the section "Deleting a Generic Target from the Preferences" on page 6-12 at the end of this tutorial.**

Creating a Configuration File from a PolySpace Project Model File

In this section...

Why You Must Have a Configuration File

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

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

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

3. Save the configuration file.

Opening the Project Model File

To open the project model file:

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

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

2 Navigate to the `polyspace_project` folder.

3 In **File of type:**, select **Project Model (*.ppm) files** from the menu.

4 Select `target_training.ppm` and click **Open**.

A message appears telling you that this project has no source files.

5 Click **OK** to close the message dialog box.

Entering Additional Required Information

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

To complete the required project information:

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

Note For more information about adding source files and include folders to a project, see "Creating a New Project to Verify a Class in the Training C++ File" on page 2-9.

Saving the Configuration File

To save the configuration file:

1 Select **File > Save project**.

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

2 Navigate to the `polyspace_project` folder.

3 In **Session identifier**, enter `training2`.

4 Leave the default type as `*.cfg`.

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

Note Your preferences still include the generic target `target1` . Complete "Deleting a Generic Target from the Preferences" on page 6-12 to delete this generic target from your preferences.

Deleting a Generic Target from the Preferences

In this section...

Understanding the Generic Targets Preference

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

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

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

Deleting the Generic Target Added in This Tutorial

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

1. In **Analysis options**, expand **Target/Compilation**.
2. If **Target processor type** is **target1**, change it to **sparc** (You cannot delete a generic target if it is the target processor type for the current project.)
3. Select **Edit > Preferences**.

   The **Preferences** dialog box appears.
4. Select the **Generic targets** tab.
5. Select `target1` from the list.
6. Click **Remove**.
7. Click **OK** to apply the change and close the dialog box.

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

Index

R

S

T

U