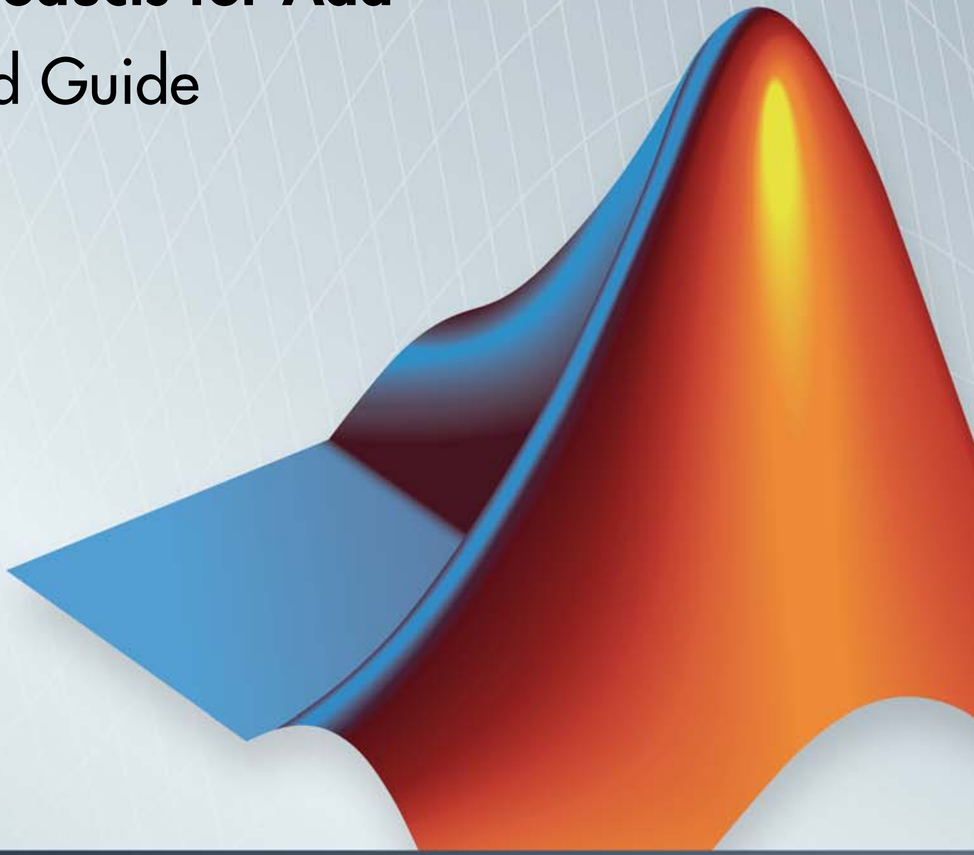
Polyspace® Products for Ada

Getting Started Guide

R2014**a**



MathWorks®

How to Contact MathWorks

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

@

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

☎ 508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Polyspace® Products for Ada Getting Started Guide

Trademarks

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

Patents

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

Revision History

March 2008	First printing	Revised for Version 5.1 (Release 2008a)
October 2008	Second printing	Revised for Version 5.2 (Release 2008b)
March 2009	Third printing	Revised for Version 5.3 (Release 2009a)
September 2009	Online only	Revised for Version 5.4 (Release 2009b)
March 2010	Online only	Revised for Version 5.5 (Release 2010a)
September 2010	Online only	Revised for Version 6.0 (Release 2010b)
April 2011	Fourth printing	Revised for Version 6.1 (Release 2011a)
September 2011	Online only	Revised for Version 6.2 (Release 2011b)
March 2012	Online only	Revised for Version 6.3 (Release 2012a)
September 2012	Online only	Revised for Version 6.4 (Release 2012b)
March 2013	Online only	Revised for Version 6.5 (Release 2013a)
September 2013	Online only	Revised for Version 6.6 (Release 2013b)
March 2014	Online Only	Revised for Version 6.7 (Release 2014a)

Contents

Introduction to Polyspace Products for Verifying Ada Code

1

Setting Up Polyspace Project

2

Running Verification

3

Reviewing Verification Results

4

Code Verification in IBM Rational Rhapsody Environment

5

Introduction to Polyspace Products for Verifying Ada Code

Product Overview

In this section...
"Polyspace® Client™ for Ada" on page 1-2
"Polyspace® Server™ for Ada" on page 1-2

Polyspace Client for Ada

Prove the absence of run-time errors in source code

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

Key Features

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

Polyspace Server for Ada

Perform code verification on computer clusters and publish metrics

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

Key Features

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

Polyspace Verification

In this section...

Overview of Polyspace Verification

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

To verify the source code, you set up verification parameters in a project, run the verification, and review the results. A graphical user interface helps you to efficiently review verification results. The software assigns a color to operations in the source code as follows

- **Green** – Indicates that an operation is proven to not have certain kinds of error.
- **Red** – Indicates that an operation is proven to have at least one error.
- **Gray** – Indicates unreachable code.
- **Orange** – Indicates that the operation can have an error along some execution paths.

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

The Value of Polyspace Verification

Polyspace verification can help you to:

- "Enhance Software Reliability" on page 1-5
- "Decrease Development Time" on page 1-5
- "Improve Development Process" on page 1-6

Enhance Software Reliability

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

Polyspace software can:

- Prove that your code has certain kinds of errors.
- Prove that your code does not have certain kinds of errors.
- Identify unreachable code.
- Identify code that can have an error along some execution paths.

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

Decrease Development Time

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

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

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

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

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

Improve Development Process

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

Polyspace verification software supports code verification throughout the development process:

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

Product Components

In this section...

Polyspace Verification Environment

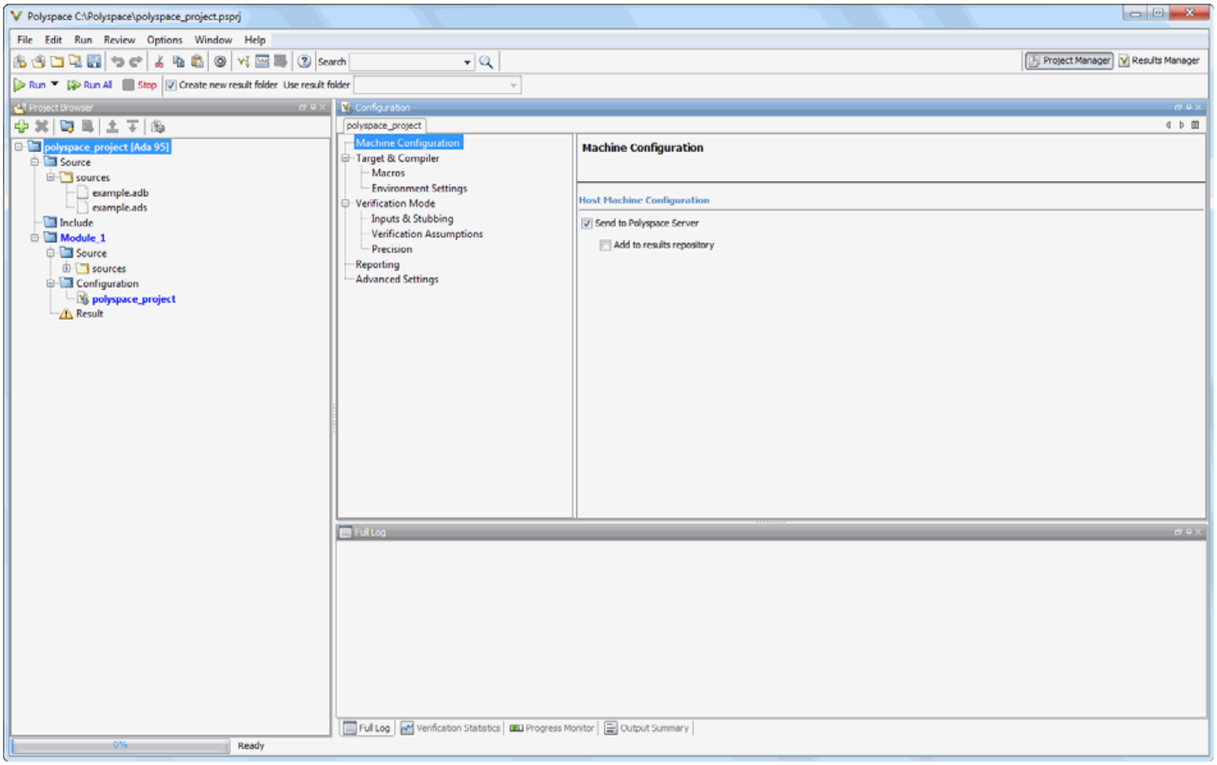
The Polyspace verification environment (PVE) is the graphical user interface of the Polyspace Client for Ada software. You use the Polyspace verification environment to create Polyspace projects, launch verifications, and review verification results.

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

- "Project Manager Perspective" on page 1-7
- "Results Manager Perspective" on page 1-8

Project Manager Perspective

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



Machine Configuration

Host Machine Configuration

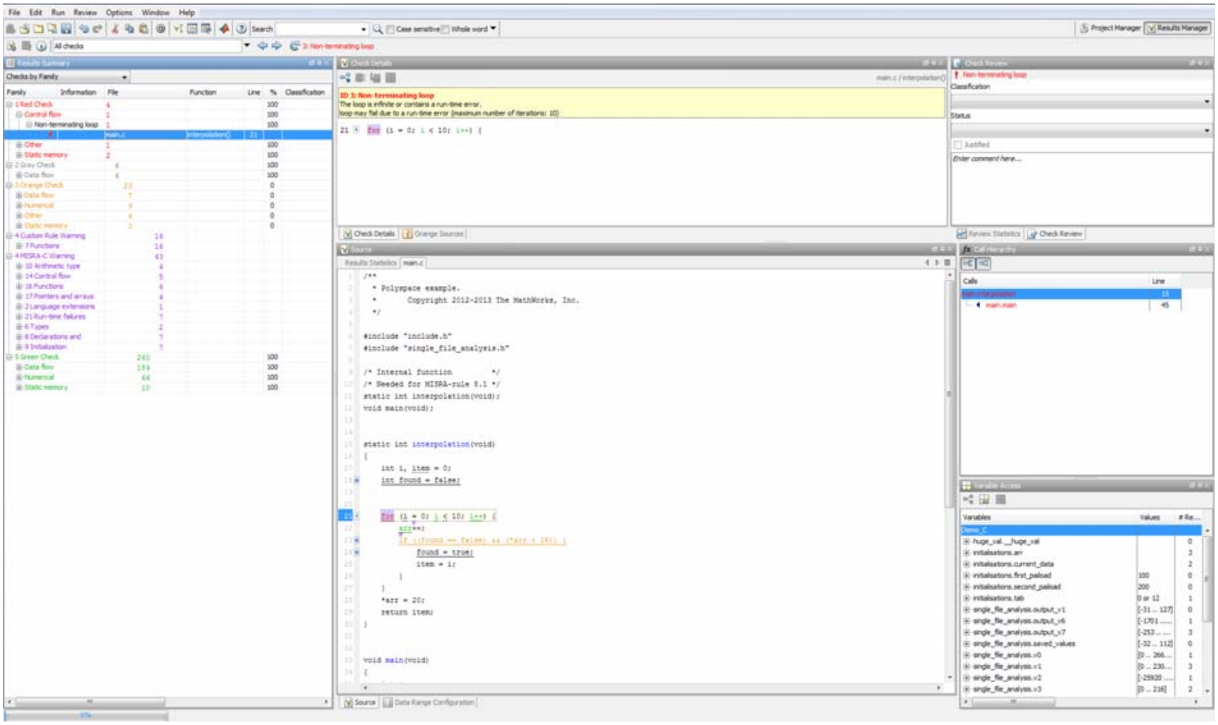
☑ Send to Polyspace Server

□ Add to results repository

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

Results Manager Perspective

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



Family	Information	File	Function	Line	%	Classification
1 Red Check		4			100	
Control flow		1			100	
Non-terminating loop		1			100	
		main.c	interpolation()	21		
Other		1			100	
Static memory		2			100	
2 Gray Check		6			100	
Data flow		6			100	
3 Orange Check		21			0	
Data flow		7			0	
Numerical		9			0	
Other		4			0	
Static memory		1			0	
4 Custom Rule Warning		16				
7 Functions		16				
4 MISRA-C Warning		43				
10 Arithmetic type		4				
14 Control flow		5				
16 Functions		6				
17 Pointers and arrays		4				
2 Language extensions		1				
21 Run-time failures		7				
6 Types		2				
8 Declarations and		7				
9 Initialization		7				
5 Green Check		240			100	
Data flow		184			100	
Numerical		46			100	
Static memory		10			100	

main.c / interpolation()

ID 3: Non-terminating loop
The loop is infinite or contains a run-time error.
loop may fail due to a run-time error (maximum number of iterations: 10)

```
21    for (i = 0; i < 10; i++) {
```

! Non-terminating loop
Classification
Status
□ Justified
Enter comment here...

```
/**
 * Polyspace example.
 *       Copyright 2012-2013 The MathWorks, Inc.
 */

#include "include.h"
#include "single_file_analysis.h"

/* Internal function        */
/* Needed for MISRA-rule 8.1 */
static int interpolation(void);
void main(void);


static int interpolation(void)
{
    int i, item = 0;
    int found = false;


    for (i = 0; i < 10; i++) {
        arr++;
        if ((found == false) && (*arr > 10)) {
            found = true;
            item = i;
        }
    }
    *arr = 20;
    return item;
}


void main(void)
{
```

Calls	Line
[illegible]	15
main.main	45

Variables	Values	# Re...
Demo_C		
huge_val.__huge_val		0
initialisations.arr		3
initialisations.current_data		2
initialisations.first_paload	100	0
initialisations.second_paload	200	0
initialisations.tab	0 or 12	1
single_file_analysis.output_v1	[-31 ... 127]	0
single_file_analysis.output_v6	[-1701	1
single_file_analysis.output_v7	[-253	3
single_file_analysis.saved_values	[-32 ... 112]	0
single_file_analysis.v0	[0 ... 266....	1
single_file_analysis.v1	[0 ... 230....	3
single_file_analysis.v2	[-25920	1
single_file_analysis.v3	[0 ... 216]	2

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

Other Polyspace Components

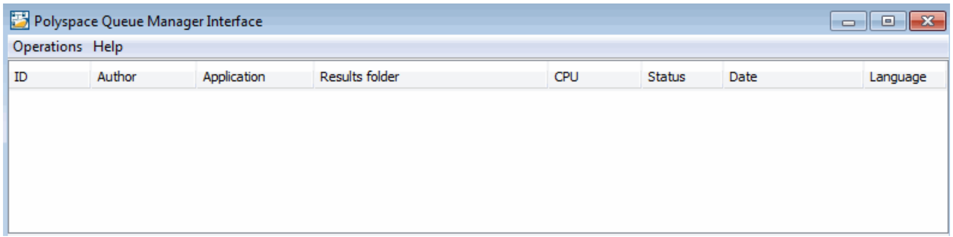
In addition to the Polyspace verification environment, Polyspace products provide other components to manage verifications, improve productivity, and track software quality. These components include:

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

Polyspace Queue Manager Interface (Polyspace Spooler)

The Polyspace Queue Manager (also called the Polyspace Spooler) is the graphical user interface of the Polyspace Server for Ada software. You use the

Polyspace Queue Manager Interface to move jobs within the queue, remove jobs, monitor the progress of individual verifications, and download results.



ID	Author	Application	Results folder	CPU	Status	Date	Language

You use the Polyspace Queue Manager in the tutorial “Starting Server Verification from Project Manager” on page 3-5.

Polyspace Metrics Web Interface

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

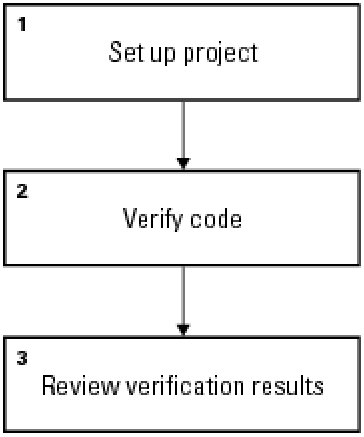
For information on using Polyspace Metrics, see “Software Quality with Polyspace Metrics”.

Working with Polyspace Software

In this section...

Basic Workflow

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



In this workflow, you:

1. Use the Project Manager perspective to set up a project file.

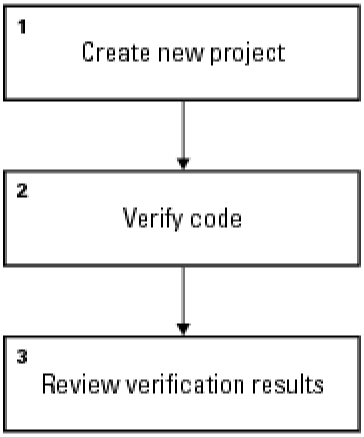
2. Verify code on a server or client.

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

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

The Workflow in This Guide

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



In this workflow, you will:

1. Create a new project that you can use for the other steps in the workflow. See the tutorial "Setting Up a Polyspace Project" on page 2-2.

2. Verify a single Ada source code package.

   See the tutorial "Running a Verification" on page 3-2. In this tutorial, you will use the Project Manager to run a verification of the package on a server and a client.

3. Review the verification results. See the tutorial "Reviewing Verification Results" on page 4-2.

Learning More

In this section...

Product Help

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

MathWorks Online

For additional information and support, see:

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

Related Products

In this section…

Polyspace Code Prover

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

```
http://www.mathworks.com/products/polyspace-code-prover
```

Polyspace Bug Finder

For information about Polyspace products that analyze C/C++ code to find possible defects, see the following:

```
http://www.mathworks.com/products/polyspace-bug-finder
```

2

Setting Up Polyspace Project

Setting Up a Polyspace Project

In this section...

About This Tutorial

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

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

Creating a New Project

What Is a Project?

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

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

- Verification results

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

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

Preparing Project Folders

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

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

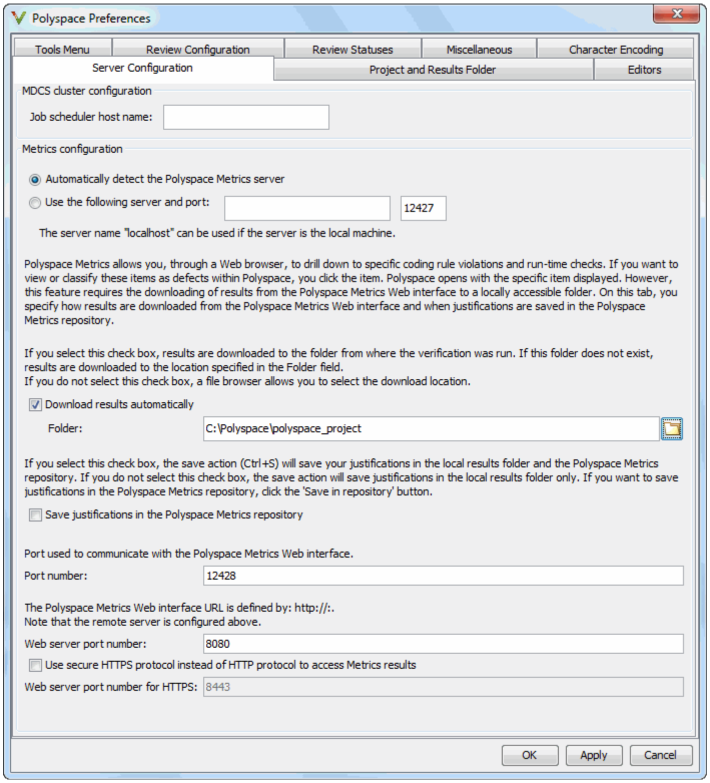
For this tutorial, prepare a project folder as follows:

1. Create a project folder named `polyspace_project`.
2. Open `polyspace_project`, and create the following folders:
   - `sources`
   - `includes`
   - `results`
3. From *`Polyspace_Install`*`\polyspace\examples\ada\Demo_Ada_Single-File\sources`, copy the files `example.adb` and `example.ads` to `polyspace_project\sources`
4. From*`Polyspace_Install`*`\polyspace\examples\ada\Demo_Ada_Single-File\source` copy all files to `polyspace_project\includes`.

Opening the Polyspace Verification Environment

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

To open the Polyspace verification environment, double-click the Polyspace icon Polyspace for Ada R2013b.



MDCS cluster configuration

Job scheduler host name:

Metrics configuration

● Automatically detect the Polyspace Metrics server

○ Use the following server and port: 12427

The server name "localhost" can be used if the server is the local machine.

Polyspace Metrics allows you, through a Web browser, to drill down to specific coding rule violations and run-time checks. If you want to view or classify these items as defects within Polyspace, you click the item. Polyspace opens with the specific item displayed. However, this feature requires the downloading of results from the Polyspace Metrics Web interface to a locally accessible folder. On this tab, you specify how results are downloaded from the Polyspace Metrics Web interface and when justifications are saved in the Polyspace Metrics repository.

If you select this check box, results are downloaded to the folder from where the verification was run. If this folder does not exist, results are downloaded to the location specified in the Folder field.
If you do not select this check box, a file browser allows you to select the download location.

☑ Download results automatically

Folder: C:\Polyspace\polyspace_project

If you select this check box, the save action (Ctrl+S) will save your justifications in the local results folder and the Polyspace Metrics repository. If you do not select this check box, the save action will save justifications in the local results folder only. If you want to save justifications in the Polyspace Metrics repository, click the 'Save in repository' button.

□ Save justifications in the Polyspace Metrics repository

Port used to communicate with the Polyspace Metrics Web interface.

Port number: 12428

The Polyspace Metrics Web interface URL is defined by: http://:.
Note that the remote server is configured above.

Web server port number: 8080

□ Use secure HTTPS protocol instead of HTTP protocol to access Metrics results

Web server port number for HTTPS: 8443

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

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

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

Creating a New Project to Verify an Ada Package

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

You create a new project by:

- "Opening a New project" on page 2-5
- "Specifying the Source Files and Include Folders" on page 2-7
- "Specifying the Analysis Options" on page 2-8
- "Specifying Source Files to Verify" on page 2-9
- "Saving the Project" on page 2-10

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

1 Select **File > New Project**. The Project - Properties dialog box opens.

2 In the **Project name** field, enter `example_project`.

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

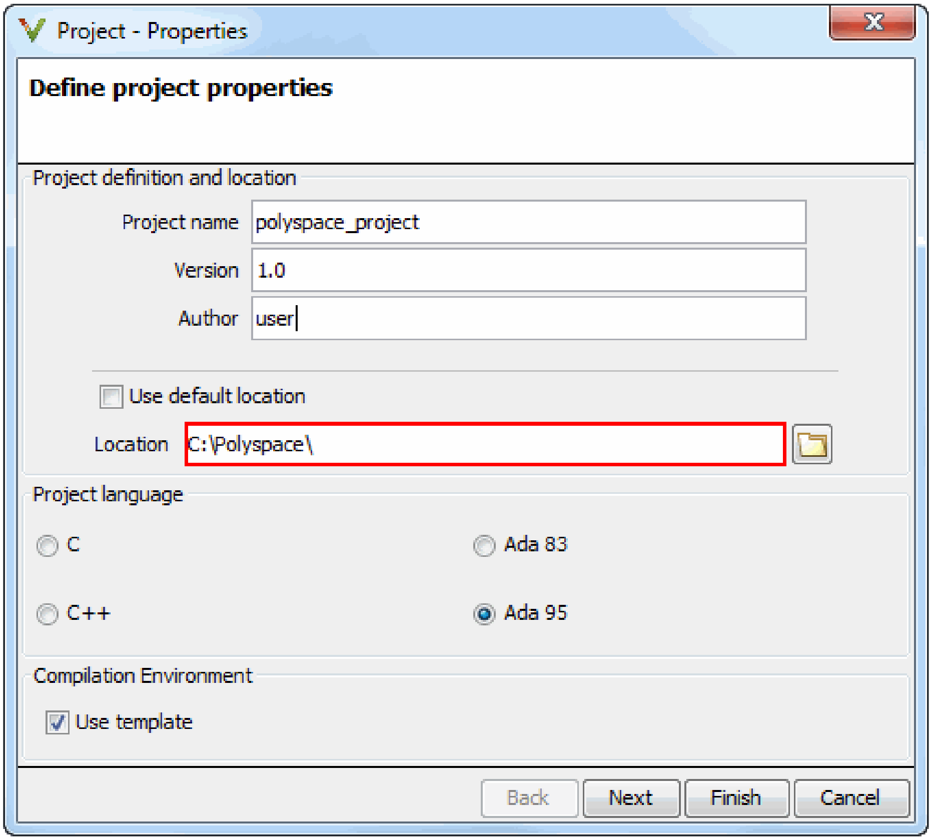
Note In this tutorial, you change the default location to the project folder that you created in "Preparing Project Folders" on page 2-3. Changing the default location makes it easier to specify source files and include folders.

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

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

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

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



Project - Properties

Define project properties

Project definition and location

Project name: polyspace_project

Version: 1.0

Author: user

□ Use default location

Location: C:\Polyspace\

Project language

○ C ○ Ada 83

○ C++ ● Ada 95

Compilation Environment

☑ Use template

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

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

1 In the **Project Browser**, select the Source folder.

2 On the Project Browser toolbar, click the icon ➕. The Project - Add Source Files and Include Folders dialog box opens.

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

4 Select the folder `sources`. Then click **Add Source Files**.

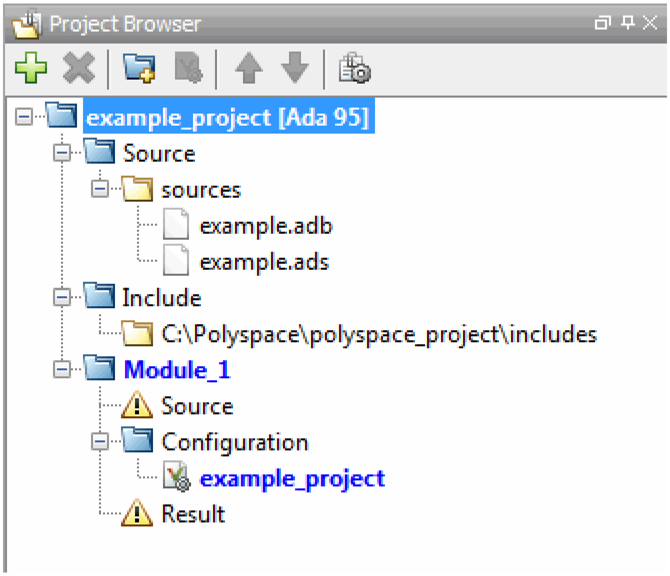
   The files, `example.adb` and `example.ads`, appear in the Source tree for `example_project`.

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

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

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

The Project Browser now looks like the following graphic.



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

To specify the analysis options for this tutorial:

1 In the Project Manager perspective, select the **Configuration > Target & Compiler** pane.

2 From the **Target operating system** drop-down list, select `no-predefined-OS`.

3 Use the default values for other options.

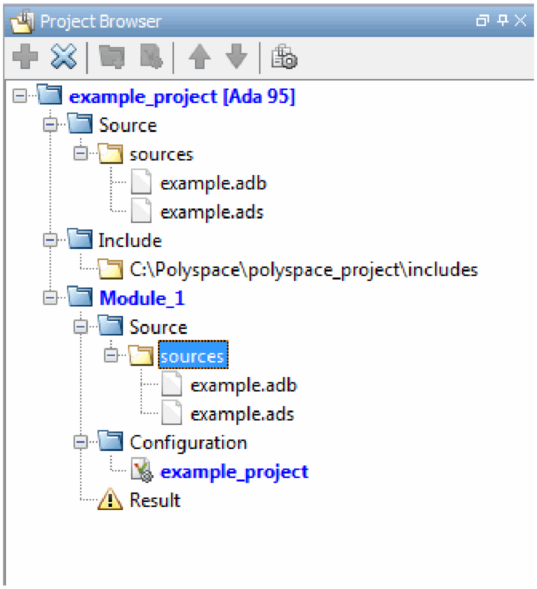
Specifying Source Files to Verify. Before you can start a verification, you must specify the files in the project that you want to verify. In `example_project`, there are two files to verify.

To specify source files for a verification:

1 In the Project Browser Source tree, right-click the folder **example_project[Ada 95] > Source > sources**, which contains the source files `example.adb` and `example.ads`.

2 From the context menu, select **Copy Source File to > Module_1.**

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



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

3

Running Verification

Running a Verification

In this section...

About This Tutorial

Overview

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

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

Use	For
Server	• Shorter verification time • Large files (more than 800 lines of code including comments)
Client	• When the server is busy • Small files **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.

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

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

Before You Start

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

Preparing for Verification

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

Opening the Project

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

To open `example_project.psprj`:

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

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

Specifying Source Files to Verify

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

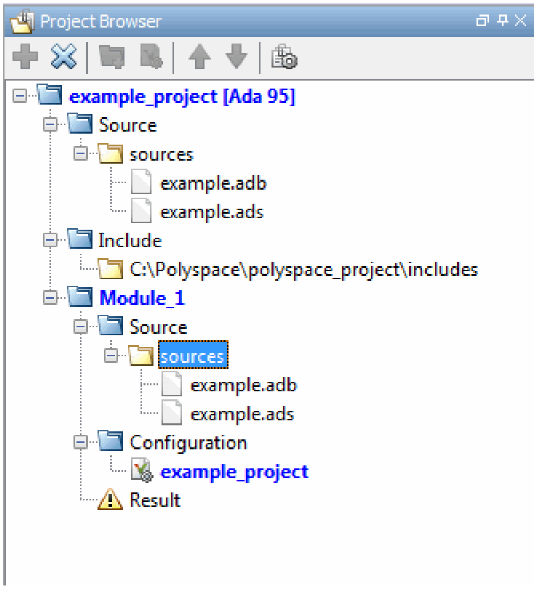
Therefore, before you can launch a verification, you must specify which files in your project you want to verify. In the `example_project` used in this tutorial, there is only one file to verify.

To copy source files to a module:

1. In the Project Browser Source tree, right-click the folder **example_project[Ada 95] > Source > sources**, which contains the source files `example.adb` and `example.ads`.

2. From the context menu, select **Copy Source File to > Module_(1).**

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



Starting Server Verification from Project Manager

- "Starting the Verification" on page 3-5
- "Monitoring the Progress of the Verification" on page 3-6
- "Removing Verification Results from the Server" on page 3-9
- "Troubleshooting a Failed Verification" on page 3-10

Starting the Verification

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

To start a verification that runs on a server:

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

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

   The verification has three main phases:

   a. Checking syntax and semantics (the compile phase). Because Polyspace software is independent of a particular Ada compiler, this phase shows whether your code is portable, maintainable, and complies with Ada standards.

   b. Generating a main subprogram if the option, **Configuration > Verification Mode > Verify module** is selected. For more information about generating a main, see "Verify module".

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

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

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

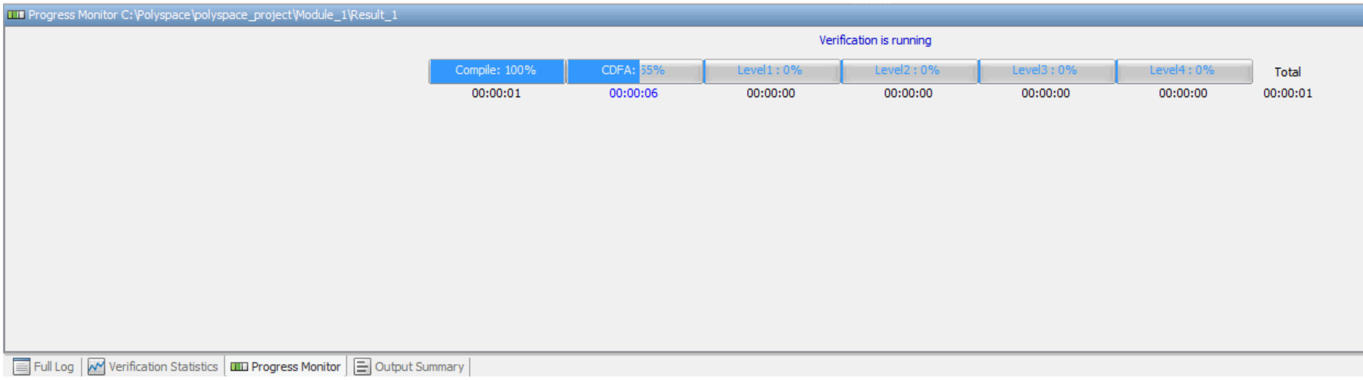
4 For information on a message in the log, click the message.

Monitoring the Progress of the Verification

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

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

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



Progress Monitor C:\Polyspace\polyspace_project\Module_1\Result_1

Verification is running

```
Compile: 100%   CDFA: 55%   Level1 : 0%   Level2 : 0%   Level3 : 0%   Level4 : 0%   Total
  00:00:01       00:00:06    00:00:00      00:00:00      00:00:00      00:00:00    00:00:01
```

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

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

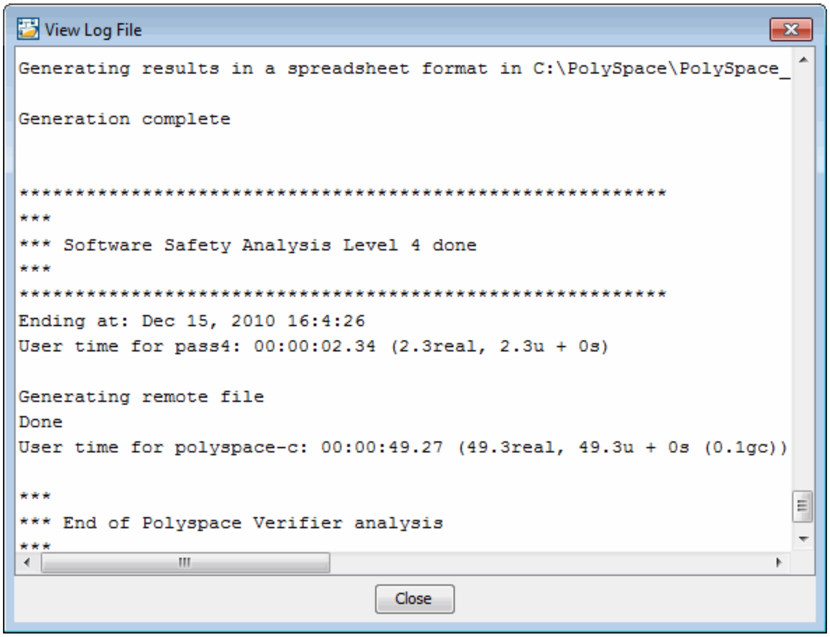
- Click the **Output Summary** tab to display compile phase messages and errors. You can search the log by entering search terms in the **Search** box and clicking the left arrow to search backward or the right arrow to search forward.
- Click the **Verification Statistics** tab to display statistics,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

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

To monitor the verification of `example_project.psprj`:

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

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



```
Generating results in a spreadsheet format in C:\PolySpace\PolySpace_

Generation complete


**********************************************************
***
*** Software Safety Analysis Level 4 done
***
**********************************************************
Ending at: Dec 15, 2010 16:4:26
User time for pass4: 00:00:02.34 (2.3real, 2.3u + 0s)

Generating remote file
Done
User time for polyspace-c: 00:00:49.27 (49.3real, 49.3u + 0s (0.1gc))

***
*** End of Polyspace Verifier analysis
***
```

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

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

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

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

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

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

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

6 Wait for the verification to finish.

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

Removing Verification Results from the Server

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

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

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

To remove your results from the server:

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

2 Click **Yes**.

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

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

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

Troubleshooting a Failed Verification

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

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

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

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

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

You can:

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

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

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

For information on how to specify the location of include files, see "Creating a New Project to Verify an Ada Package" on page 2-5.

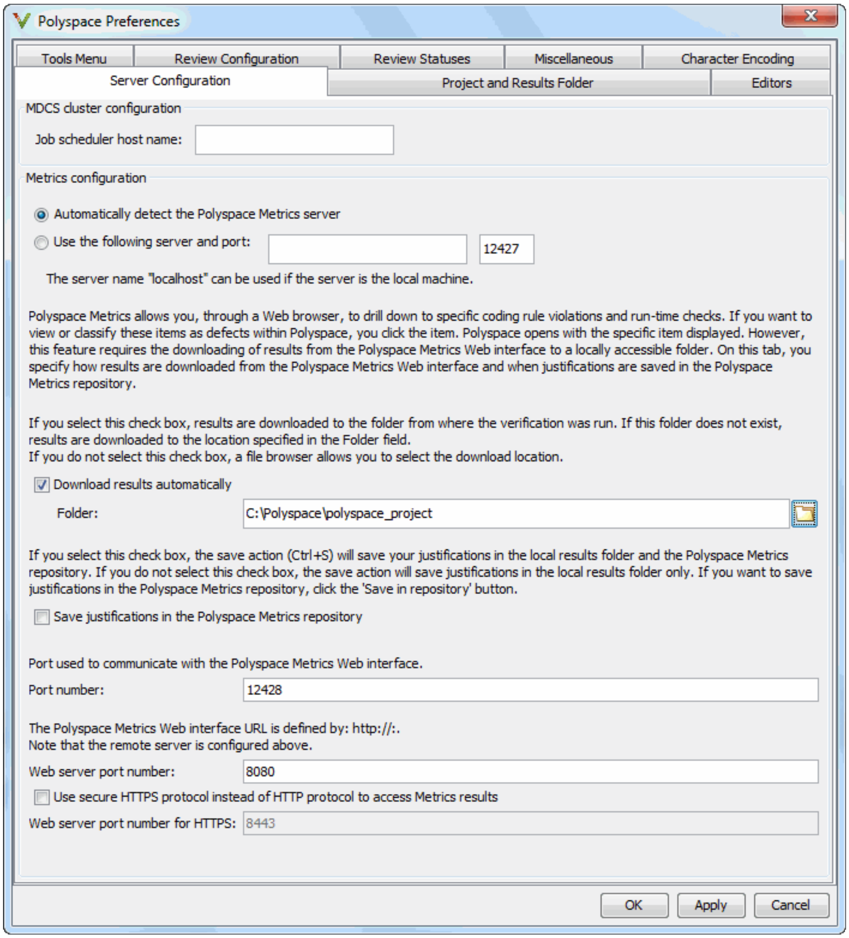
Polyspace Software Cannot Find the Server. If you see the following message in the log, Polyspace software cannot find the server.

```
Error:  Unknown host :
```

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

1. Select **Options > Preferences**.

2. Select the **Server Configuration** tab.



MDCS cluster configuration

Job scheduler host name:

Metrics configuration

● Automatically detect the Polyspace Metrics server

○ Use the following server and port: 12427

The server name "localhost" can be used if the server is the local machine.

Polyspace Metrics allows you, through a Web browser, to drill down to specific coding rule violations and run-time checks. If you want to view or classify these items as defects within Polyspace, you click the item. Polyspace opens with the specific item displayed. However, this feature requires the downloading of results from the Polyspace Metrics Web interface to a locally accessible folder. On this tab, you specify how results are downloaded from the Polyspace Metrics Web interface and when justifications are saved in the Polyspace Metrics repository.

If you select this check box, results are downloaded to the folder from where the verification was run. If this folder does not exist, results are downloaded to the location specified in the Folder field.
If you do not select this check box, a file browser allows you to select the download location.

☑ Download results automatically

Folder: C:\Polyspace\polyspace_project

If you select this check box, the save action (Ctrl+S) will save your justifications in the local results folder and the Polyspace Metrics repository. If you do not select this check box, the save action will save justifications in the local results folder only. If you want to save justifications in the Polyspace Metrics repository, click the 'Save in repository' button.

□ Save justifications in the Polyspace Metrics repository

Port used to communicate with the Polyspace Metrics Web interface.

Port number: 12428

The Polyspace Metrics Web interface URL is defined by: http://:.
Note that the remote server is configured above.

Web server port number: 8080

□ Use secure HTTPS protocol instead of HTTP protocol to access Metrics results

Web server port number for HTTPS: 8443

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

Starting Client Verification from Project Manager

- "Starting the Verification" on page 3-13
- "Monitoring the Progress of the Verification" on page 3-14

- “Completing Verification” on page 3-15
- “Stopping the Verification” on page 3-15

Starting the Verification

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

Note Because a verification on a client can process only a limited number of variable assignments and function calls, the source code should not have more than 800 lines of code.

To start a verification that runs on a client:

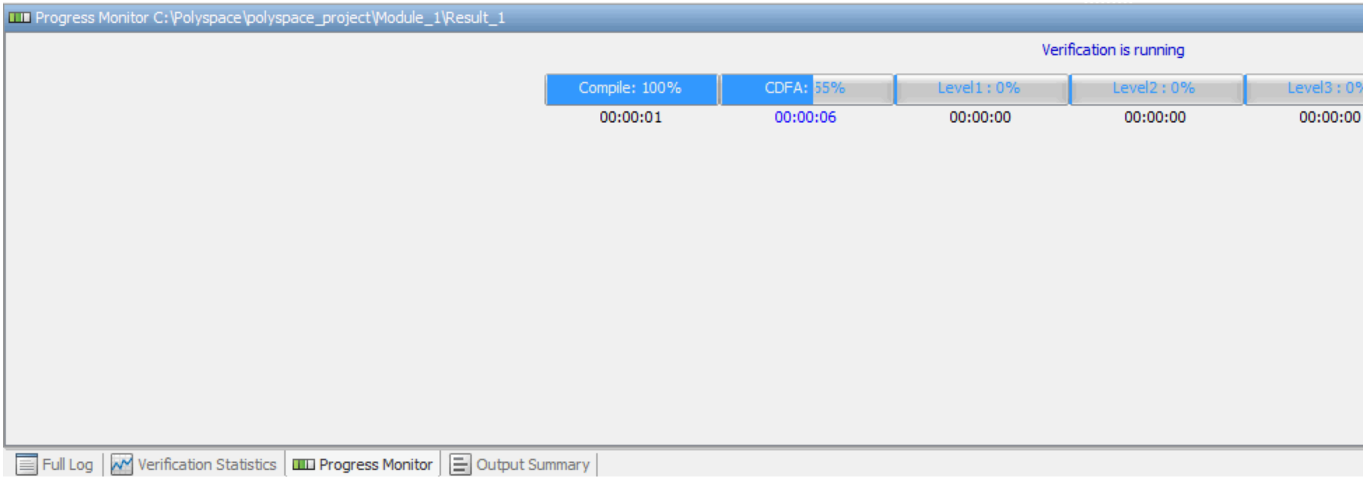
1. Open the project `example_project.psprj`. For information about opening a project, see “Preparing for Verification” on page 3-3.
2. In the Project Manager perspective, select the **Configuration > Machine Configuration** pane.
3. Clear the **Send to Polyspace Server** check box.
4. On the Project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

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

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

Monitoring the Progress of the Verification

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



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

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

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

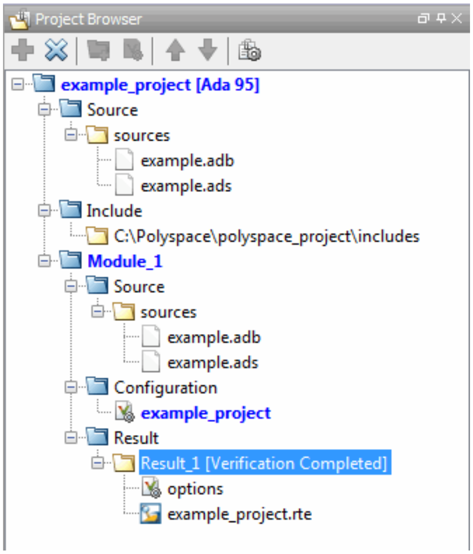
2. Click the **Verification Statistics** tab to display statistics, such as analysis options, stubbed functions, and the verification checks performed.

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

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

Completing Verification

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



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

Stopping the Verification

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

To stop a verification:

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

A warning dialog box opens, requesting confirmation.

2 Click **Yes**. The verification stops.

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

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

Reviewing Verification Results

Reviewing Verification Results

In this section...

About This Tutorial

Overview

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

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

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

Prerequisites

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

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

Opening Verification Results

- "Opening the Results Manager Perspective" on page 4-3
- "Opening Verification Results" on page 4-3

Opening the Results Manager Perspective

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

Opening Verification Results

To open the verification results:

1. In the Polyspace verification environment, select **File > Open Result**.

   The Open results dialog box opens.

2. Navigate to the results folder:

   `polyspace_project\Module_1\Result_1`.

3. Select the file `RTE_px_example_project_LAST_RESULTS.rte`.

4. Click **Open**. The results appear in the Results Manager perspective.

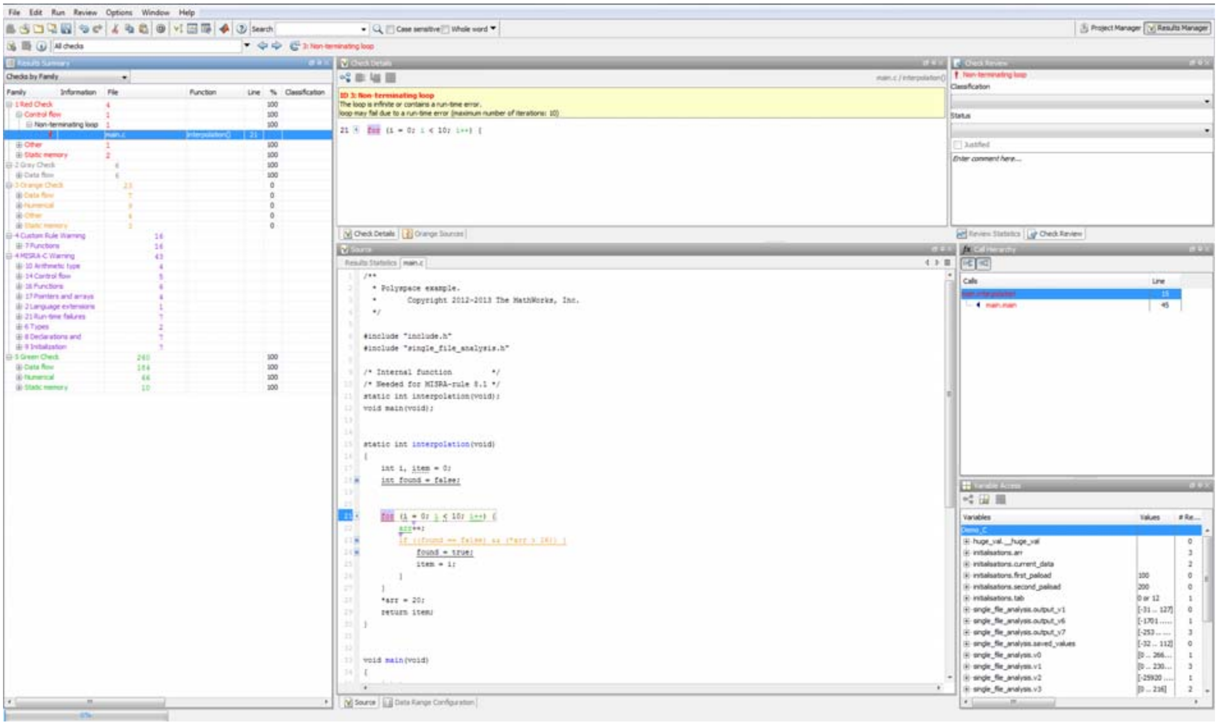
Note You can also open results from the Project Manager perspective by double-clicking the results file in the Project Browser.

Exploring Results Manager Perspective

- "Overview" on page 4-4
- "Results Summary" on page 4-5

Overview

The Results Manager perspective looks like the following graphic.



Family	Information	File	Function	Line	%	Classification
1 Red Check	4				100	
Control flow	1				100	
Non-terminating loop	1				100	
		main.c	interpolation()	21		
Other	1				100	
Static memory	2				100	
2 Gray Check	6				100	
Data flow	6				100	
3 Orange Check	23				0	
Data flow	7				0	
Numerical	9				0	
Other	4				0	
Static memory	3				0	
4 Custom Rule Warning	16					
7 Functions	16					
4 MISRA-C Warning	43					
10 Arithmetic type	4					
14 Control flow	5					
16 Functions	6					
17 Pointers and arrays	4					
2 Language extensions	1					
21 Run-time failures	7					
6 Types	2					
8 Declarations and	7					
9 Initialization	7					
5 Green Check	240				100	
Data flow	184				100	
Numerical	46				100	
Static memory	10				100	

main.c / interpolation()

ID 3: Non-terminating loop
The loop is infinite or contains a run-time error.
loop may fail due to a run-time error (maximum number of iterations: 10)

```
21  for (i = 0; i < 10; i++) {
```

Non-terminating loop
Classification
Status
□ Justified
Enter comment here....

```
/**
 * Polyspace example.
 *       Copyright 2012-2013 The MathWorks, Inc.
 */

#include "include.h"
#include "single_file_analysis.h"

/* Internal function        */
/* Needed for MISRA-rule 8.1 */
static int interpolation(void);
void main(void);


static int interpolation(void)
{
    int i, item = 0;
    int found = false;


    for (i = 0; i < 10; i++) {
        arr++;
        if ((found == false) && (*arr > 16)) {
            found = true;
            item = i;
        }
    }
    *arr = 20;
    return item;
}


void main(void)
{
```

Calls	Line
main.interpolation	15
main.main	45

Variables	Values	# Re...
Demo_C		
huge_val.__huge_val		0
initialisations.arr		3
initialisations.current_data		2
initialisations.first_paload	100	0
initialisations.second_paload	200	0
initialisations.tab	0 or 12	1
single_file_analysis.output_v1	[-31 ... 127]	0
single_file_analysis.output_v6	[-1701	1
single_file_analysis.output_v7	[-253	3
single_file_analysis.saved_values	[-32 ... 112]	0
single_file_analysis.v0	[0 ... 266....	1
single_file_analysis.v1	[0 ... 230....	3
single_file_analysis.v2	[-25920	1
single_file_analysis.v3	[0 ... 216]	2

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

This pane or view ...	Displays ...
Results Summary	List of checks (diagnostics) for each file and function in the project
Source	Source code for a selected check in the procedural entities view
Check Details	Details about the selected check
Check Review	Review information about selected check
Variable Access	Information about global variables declared in the source code
Call Hierarchy	Tree structure of function calls

You can resize or hide sections. You learn more about the Results Manager perspective later in this tutorial.

Results Summary

The **Results Summary** pane lists checks along with their attributes. To organize your check review, from the drop-down list on this pane, select one of the following options:

- `List of Checks`: Lists checks without grouping them. The checks are sorted in the following order:

  1. **Red**: Indicates code that is proven to contain an error. The check indicates that the code will fail every time it is executed.
  2. **Gray** — Indicates unreachable code.
  3. **Orange** — Indicates unproven code that might contain an error.
  4. **Green** — Indicates code that is proven to not contain an error.

- `Checks by Family`: Lists checks grouped by color. Within each color, the checks are grouped by category. For more information on the checks covered by a category, see the check reference pages.
- `Checks by Package`: Lists checks grouped by package. Within each class, the checks are grouped by prcoedures.

- `Checks by File/Function`: Lists checks grouped by file. Within each file, the checks are grouped by function.

For each check, the **Results Summary** pane contains the check attributes, listed in columns:

Attribute	Description
Family	Group to which the check belongs. For instance, if you choose the grouping `Checks by File/Function`, this column contains the name of the file and function containing the check.
Check	Description of the error
File	File containing the instruction where the check occurs
Package	Package containing the instruction where the check occurs
Line	Line number of the instruction where the check occurs.
Classification	Level of severity you have assigned to the check. The possible levels are: • `Unset` • `High` • `Medium` • `Low` • `Not a defect`

Attribute	Description
Status	Review status you have assigned to the check. The possible statuses are: • `Fix` • `Improve` • `Investigate` • `Justify with annotations` • `No action planned` • `Other` • `Restart with different options`
Justified	Check boxes showing whether you have justified the checks
Comments	Comments you have entered about the check

To show or hide a column, right-click on the column title. From the context menu, select or clear the title of the column that you want to show or hide.

Using this pane, you can:

- Navigate through the checks. For more information, see "Review and Comment Checks".
- Organize your check review using filters available in each column. For more information, see "Organize Check Review Using Filters and Groups".

Reviewing Results

View Examples of Checks

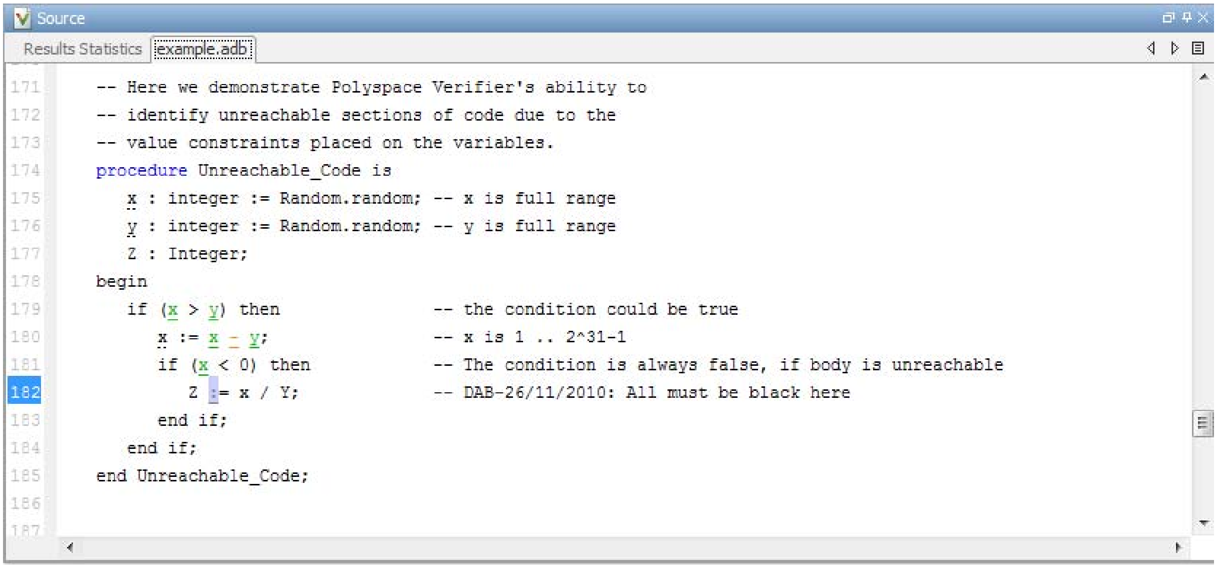
In this part of the tutorial, you learn about other types and categories of errors by reviewing the following checks in `example.adb`:

- "Example: Unreachable Code" on page 4-8
- "Example: A Function with No Errors" on page 4-9
- "Example: Division by Zero" on page 4-9

Example: Unreachable Code. Unreachable code is code that cannot be reached during normal execution of the software. The code verification software displays unreachable code in gray. In the following steps, you see an example of unreachable code.

1 On the **Results Summary** pane, expand `UNREACHABLE_CODE()` and click the gray `Unreachable code`.

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



```
-- Here we demonstrate Polyspace Verifier's ability to
-- identify unreachable sections of code due to the
-- value constraints placed on the variables.
procedure Unreachable_Code is
   x : integer := Random.random; -- x is full range
   y : integer := Random.random; -- y is full range
   Z : Integer;
begin
   if (x > y) then              -- the condition could be true
      x := x - y;               -- x is 1 .. 2^31-1
      if (x < 0) then           -- The condition is always false, if body is unreachable
         Z := x / Y;            -- DAB-26/11/2010: All must be black here
      end if;
   end if;
end Unreachable_Code;
```

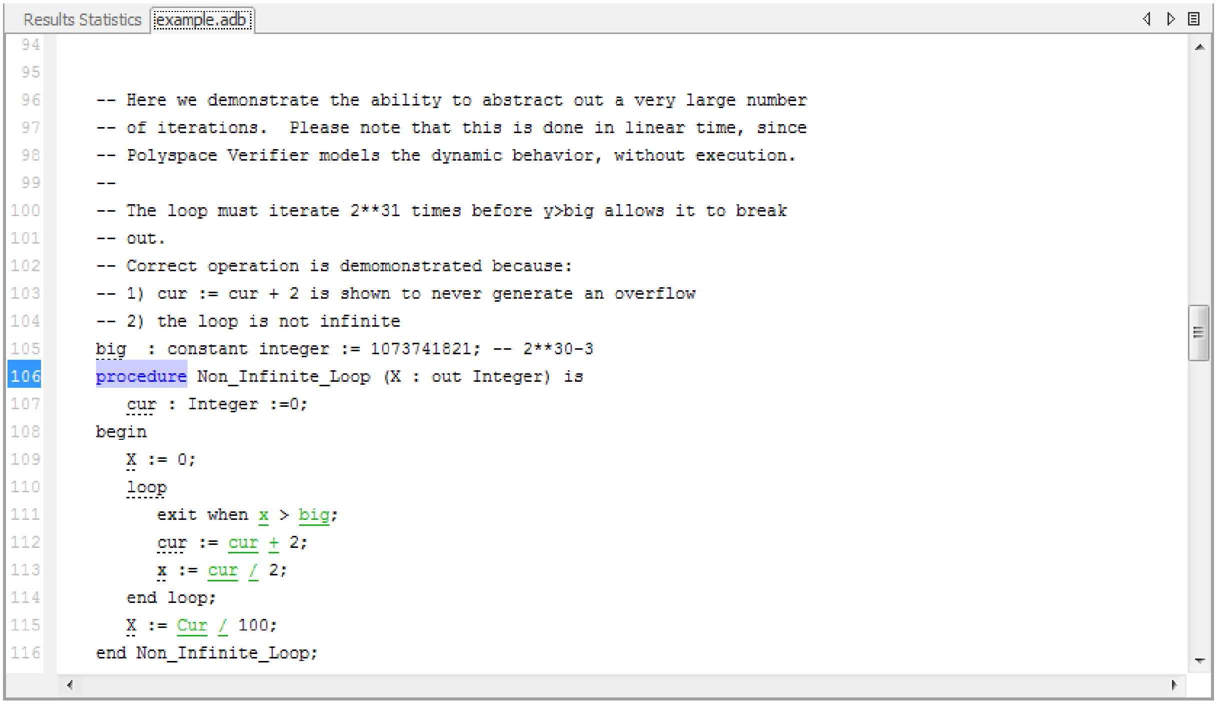
2 Examine the source code.

At line 182, the code `Z := x / Y;` cannot be reached because the condition `x < 0` is false.

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

1 On the **Results Summary** pane, click the green `NON_INFINITE_LOOP()` function.

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



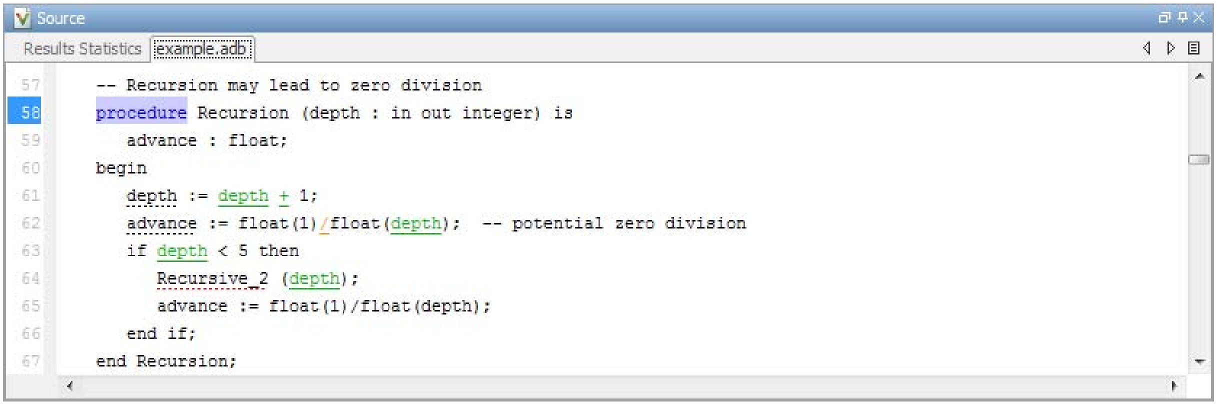
```
 94
 95
 96    -- Here we demonstrate the ability to abstract out a very large number
 97    -- of iterations.  Please note that this is done in linear time, since
 98    -- Polyspace Verifier models the dynamic behavior, without execution.
 99    --
100    -- The loop must iterate 2**31 times before y>big allows it to break
101    -- out.
102    -- Correct operation is demomonstrated because:
103    -- 1) cur := cur + 2 is shown to never generate an overflow
104    -- 2) the loop is not infinite
105    big  : constant integer := 1073741821; -- 2**30-3
106    procedure Non_Infinite_Loop (X : out Integer) is
107       cur : Integer :=0;
108    begin
109       X := 0;
110       loop
111          exit when x > big;
112          cur := cur + 2;
113          x := cur / 2;
114       end loop;
115       X := Cur / 100;
116    end Non_Infinite_Loop;
```

2 Examine the source code. The variable `cur` does not overflow because the loop at line `110` terminates before `cur` can overflow.

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

1 On the **Results Summary** pane, expand `RECURSION()`.

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



```
57     -- Recursion may lead to zero division
58     procedure Recursion (depth : in out integer) is
59        advance : float;
60     begin
61        depth := depth + 1;
62        advance := float(1)/float(depth);  -- potential zero division
63        if depth < 5 then
64           Recursive_2 (depth);
65           advance := float(1)/float(depth);
66        end if;
67     end Recursion;
```

2 Examine the `RECURSION` function.

When `RECURSION` is called with `depth` less than zero, the code at line 62 results in division by zero. The orange color indicates that this result is a potential error (depending on the value of `depth`).

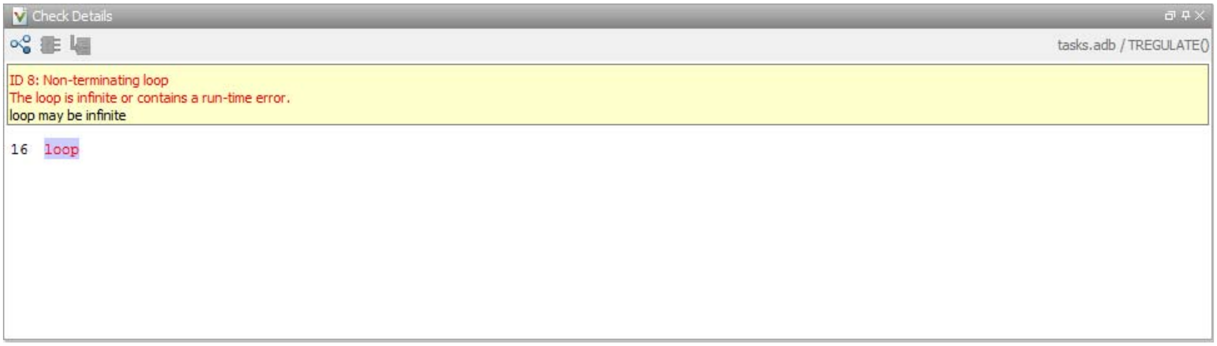
Review Checks

This example shows how to review and comment checks using the Results Manager perspective. When reviewing checks, you can assign a status to checks, and enter comments to describe the results of your review. These actions help you to track the progress of your review and avoid reviewing the same check twice.

Review and Comment Individual Check

1 On the **Results Summary** pane, select the check that you want to review.

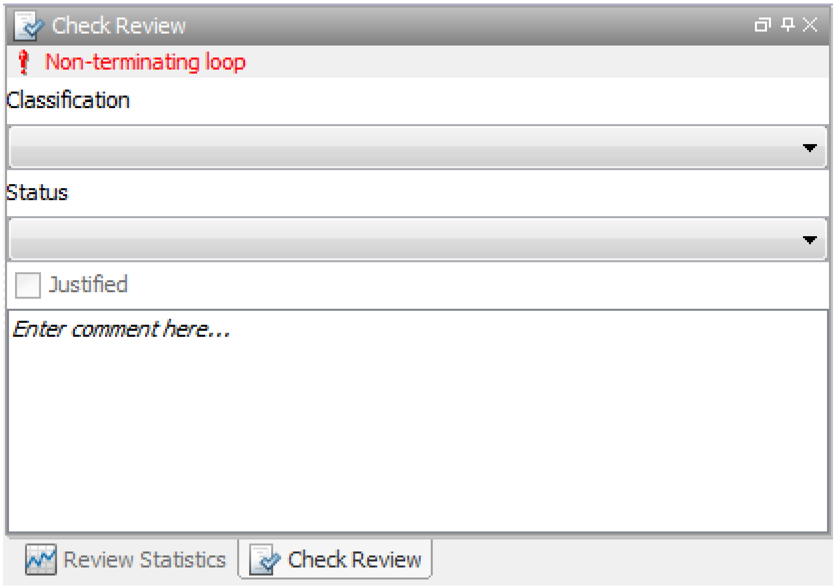
The **Check Details** pane displays information about the current check.



ID 8: Non-terminating loop
The loop is infinite or contains a run-time error.
loop may be infinite

```
16  loop
```

The **Check Review** tab displays fields where you can enter review information.



2 Select a **Classification** to describe the severity of the issue:

- `Unset`
- `High`
- `Medium`

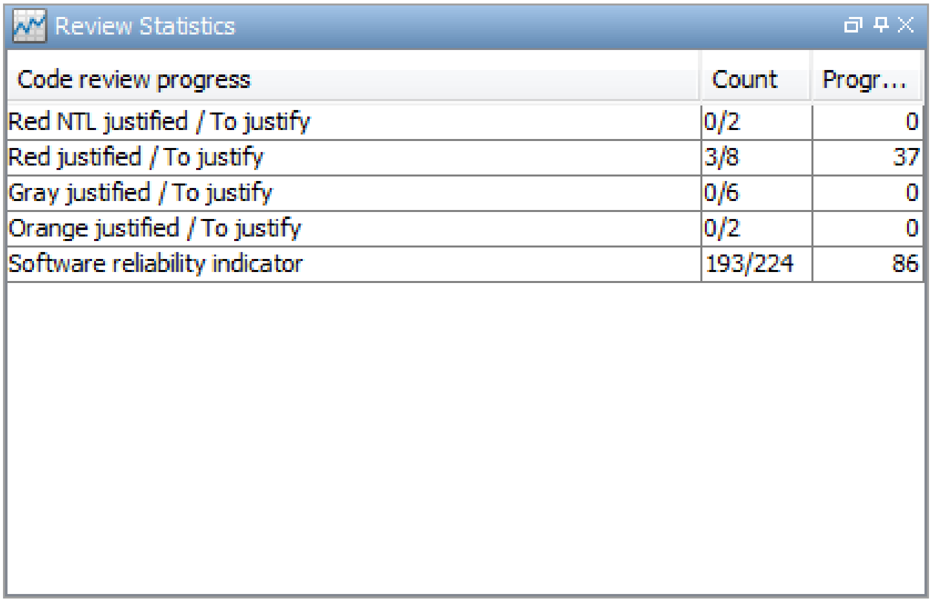
- `Low`
- `Not a defect`

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

- `Fix`
- `Improve`
- `Investigate`
- `Justify with annotations`
- `No action planned`
- `Other`
- `Restart with different options`
- `Undecided`

4 To justify the check, select one of the **Status** options, `Justify with annotations` or `No action planned`.

On the **Review Statistics** pane, the software updates the ratios of errors justified to total errors.



Code review progress	Count	Progr…
Red NTL justified / To justify	0/2	0
Red justified / To justify	3/8	37
Gray justified / To justify	0/6	0
Orange justified / To justify	0/2	0
Software reliability indicator	193/224	86

5 In the **Comment** field, enter remarks, for example, defect or justification information.

Note You can also enter the review information through the **Classification**, **Status**, and **Comment** fields on the **Results Summary** pane.

Save Review Comments

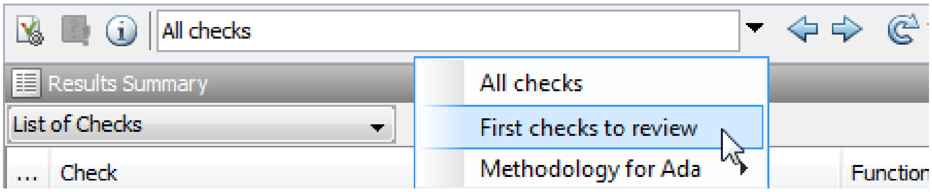
After you have reviewed your results, save your comments with the verification results. Saving your comments makes them available the next time that you open the results file, allowing you to avoid reviewing the same check twice.

To save your review comments, select **File > Save**. Your comments are saved with the verification results.

Review Checks Using Predefined Methodologies

This example shows how to incrementally review the checks using predefined review methodologies provided by Polyspace Client for Ada.

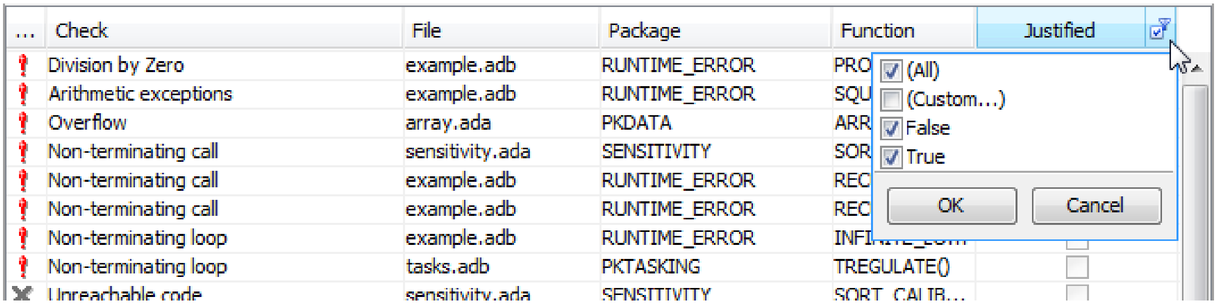
1 In the Results manager perspective, from the drop-down list above the **Results Summary** pane, select the methodology, **First checks to review**.



The **Results Summary** pane displays all red and gray checks, as well as orange checks most likely to be run-time errors. Investigate and fix the errors, set the status of the checks, and justify them.

2 Once the checks have been justified, select the methodology, **Methodology for Ada > Light**.

In the **Results Summary** pane, you can view all red and gray checks, as well as a subset of orange checks. To filter the unjustified checks, from the drop-down list beside the **Justified** column header, clear all boxes except **False** and select **OK**.



...	Check	File	Package	Function	Justified
!	Division by Zero	example.adb	RUNTIME_ERROR	PRO	
!	Arithmetic exceptions	example.adb	RUNTIME_ERROR	SQU	
!	Overflow	array.ada	PKDATA	ARR	
!	Non-terminating call	sensitivity.ada	SENSITIVITY	SOR	
!	Non-terminating call	example.adb	RUNTIME_ERROR	REC	
!	Non-terminating call	example.adb	RUNTIME_ERROR	REC	
!	Non-terminating loop	example.adb	RUNTIME_ERROR	INFI	□
!	Non-terminating loop	tasks.adb	PKTASKING	TREGULATE()	□
✗	Unreachable code	sensitivity.ada	SENSITIVITY	SORT_CALIB...	□

Investigate and fix the errors, set the status of the checks, and justify them.

3 Once the checks have been justified, to see a larger subset of orange checks, select **Methodology for Ada > Moderate**. The number of orange checks on the **Results Summary** pane increases.

To refresh the list to show unjustified checks only, reopen the drop-down list beside the **Justified** column header and select **OK**.

Investigate and fix the errors, set the status of the checks and justify them.

4 To exhaustively review all checks, select **All checks**. In addition to all orange checks, this methodology also reveals all green checks on the **Results Summary** pane.

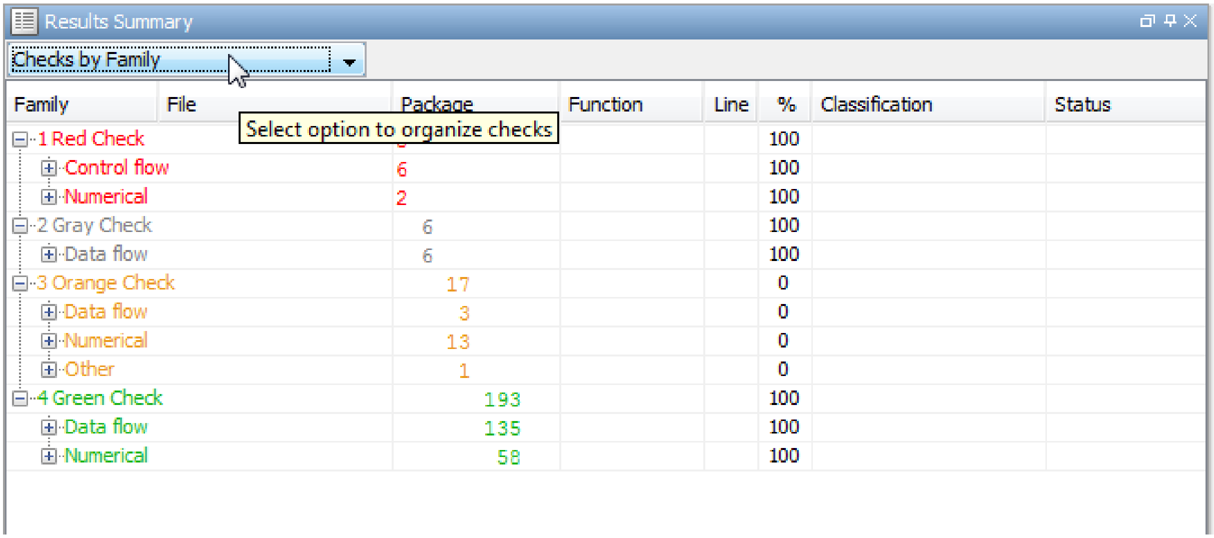
Organize Check Review Using Filters and Groups

To review checks resulting from `Non-terminating loop`:

1 Open the results file, with extension, `.rte`.

2 On the **Results Summary** pane, from the drop-down list, select `Checks by Family`.

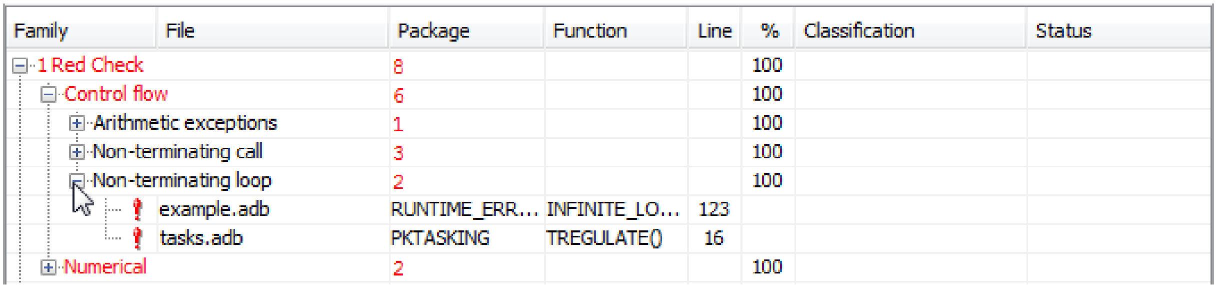
The checks are grouped by type of check.



Family	File	Package	Function	Line	%	Classification	Status
1 Red Check		[illegible]			100		
Control flow		6			100		
Numerical		2			100		
2 Gray Check		6			100		
Data flow		6			100		
3 Orange Check		17			0		
Data flow		3			0		
Numerical		13			0		
Other		1			0		
4 Green Check		193			100		
Data flow		135			100		
Numerical		58			100		

3 Under the category **1 Red Check**, expand the subcategory **Control flow**.

You see the subcategory **Non-terminating loop**.



Family	File	Package	Function	Line	%	Classification	Status
⊟ 1 Red Check		8			100		
⊟ Control flow		6			100		
⊞ Arithmetic exceptions		1			100		
⊞ Non-terminating call		3			100		
⊟ Non-terminating loop		2			100		
!	example.adb	RUNTIME_ERR…	INFINITE_LO…	123			
!	tasks.adb	PKTASKING	TREGULATE()	16			
⊞ Numerical		2			100		

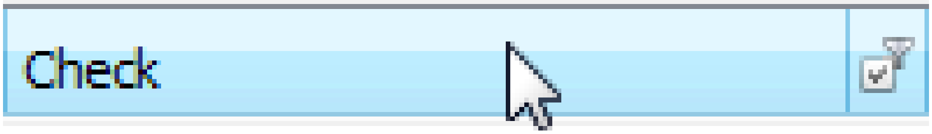
Expand **Non-terminating loop** to view red checks resulting from this error.

To see further information about a check, select it. The information appears on the **Check Details** pane.

4 To view orange checks resulting from this error, repeat step 3 for the subcategory **Control flow** under the category **3 Orange Check**.

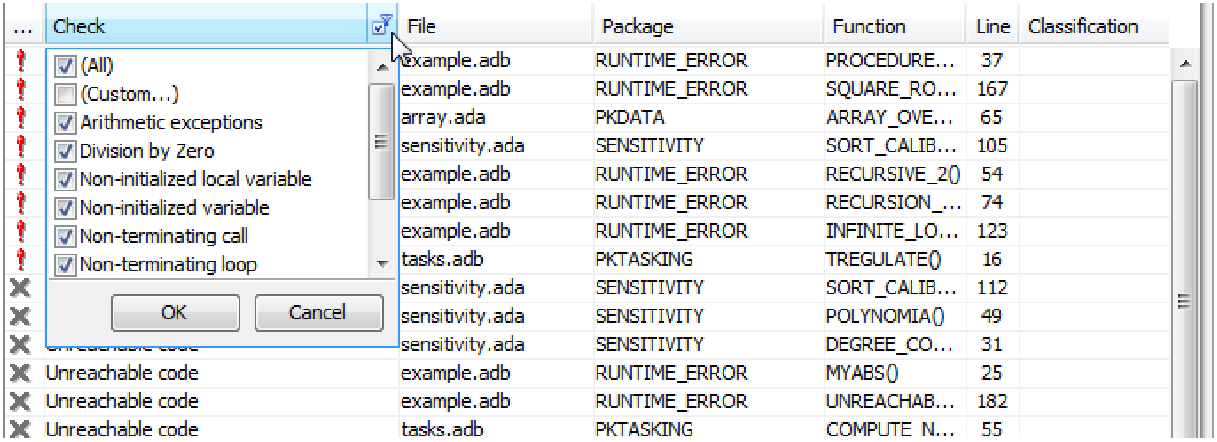
5 To view only the checks resulting from the error, `Non-terminating loop`, on the **Results Summary** pane, from the drop-down list, select `List of Checks`.

6 Place your cursor on the **Check** column head.



7 Click the filter icon.

A context menu lists the filter options available.



- ☑ (All)
- □ (Custom...)
- ☑ Arithmetic exceptions
- ☑ Division by Zero
- ☑ Non-initialized local variable
- ☑ Non-initialized variable
- ☑ Non-terminating call
- ☑ Non-terminating loop

...	Check	File	Package	Function	Line	Classification
!		example.adb	RUNTIME_ERROR	PROCEDURE...	37	
!		example.adb	RUNTIME_ERROR	SQUARE_RO...	167	
!		array.ada	PKDATA	ARRAY_OVE...	65	
!		sensitivity.ada	SENSITIVITY	SORT_CALIB...	105	
!		example.adb	RUNTIME_ERROR	RECURSIVE_2()	54	
!		example.adb	RUNTIME_ERROR	RECURSION_...	74	
!		example.adb	RUNTIME_ERROR	INFINITE_LO...	123	
!		tasks.adb	PKTASKING	TREGULATE()	16	
✗		sensitivity.ada	SENSITIVITY	SORT_CALIB...	112	
✗		sensitivity.ada	SENSITIVITY	POLYNOMIA()	49	
✗	Unreachable code	sensitivity.ada	SENSITIVITY	DEGREE_CO...	31	
✗	Unreachable code	example.adb	RUNTIME_ERROR	MYABS()	25	
✗	Unreachable code	example.adb	RUNTIME_ERROR	UNREACHAB...	182	
✗	Unreachable code	tasks.adb	PKTASKING	COMPUTE_N...	55	

8 Clear the **All** check box.

9 Scroll down to the **Non-terminating loop** check box and select it. Click **OK**.

The **Results Summary** pane displays only the checks resulting from the `Non-terminating loop` error.

Generating Reports of Verification Results

- "Polyspace Report Generator Overview" on page 4-17
- "Generating Report for example.adb" on page 4-18

Polyspace Report Generator Overview

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

The Polyspace Report Generator provides the following report templates:

- **Coding Rules** — Provides information about compliance with MISRA-C Coding Rules, as well as Polyspace configuration settings for the verification.

- **Developer** — Provides information useful to developers, including summary results, detailed lists of red, orange, and gray checks, and Polyspace configuration settings for the verification.

  Developer Review – Provides the same information as the Developer Report, but reviewed results are sorted by review classification (`High`, `Medium`, `Low`, `Not a defect`) and status, and untagged checks are sorted by file location.
- **Developer_withGreenChecks** — Provides the same content as the Developer Report, but also includes a detailed list of green checks.
- **Quality** — Provides information useful to quality engineers, including summary results, statistics about the code, graphs showing distributions of checks per file, and Polyspace configuration settings for the verification.

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

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

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

Generating Report for example.adb

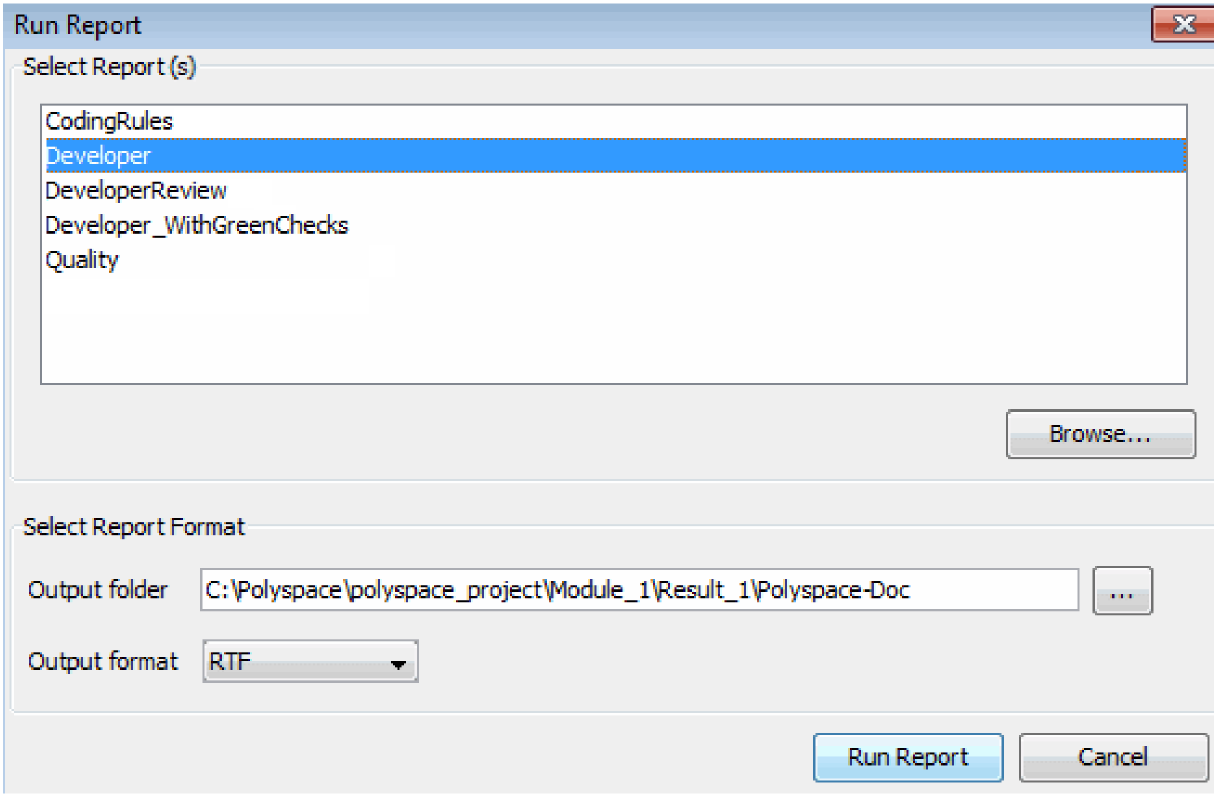
You can generate reports for verification results using the Polyspace Report Generator.

To generate a verification report:

1. Open your verification results.

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

The Run Report dialog box opens.



Output folder C:\Polyspace\polyspace_project\Module_1\Result_1\Polyspace-Doc

Output format RTF

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

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

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

6 Click **Run Report**.

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

5

Code Verification in IBM Rational Rhapsody Environment

Verify Code in IBM Rational Rhapsody Environment

In this section...

Code Verification Approach

In a collaborative Model-Driven Development (MDD) environment, software run-time errors can be produced by either design issues in the model or faulty handwritten code. You may be able to detect the flaws using code reviews and intensive testing. However, these techniques are time-consuming and expensive.

With Polyspace Products for Ada, you can verify Ada code that you generate from your IBM® Rational® Rhapsody® model. As a result, you can detect run-time errors and automatically identify model flaws quickly and early during the design process.

For information about installing and using IBM Rational Rhapsody, go to www-01.ibm.com/software/awdtools/rhapsody/.

The approach for using Polyspace Products for Ada within the IBM Rational Rhapsody MDD environment is:

- Integrate the Polyspace add-in with your Rhapsody project. See "Adding Polyspace Profile to Model" on page 5-3.

- If required, specify Polyspace configuration options in the Polyspace verification environment. See "Configuring Verification Options" on page 5-6.
- Specify the `include` path to your operating system (environment) header files and run verification. See "Running a Verification" on page 5-6 and "Monitoring a Verification" on page 5-7.
- View results, analyze errors, and locate faulty code within model. See "Viewing Polyspace Results" on page 5-7 and "Locating Faulty Code in Rhapsody Model" on page 5-8.

Adding Polyspace Profile to Model

Before you try to access Polyspace features, you must add the Polyspace profile to your model :

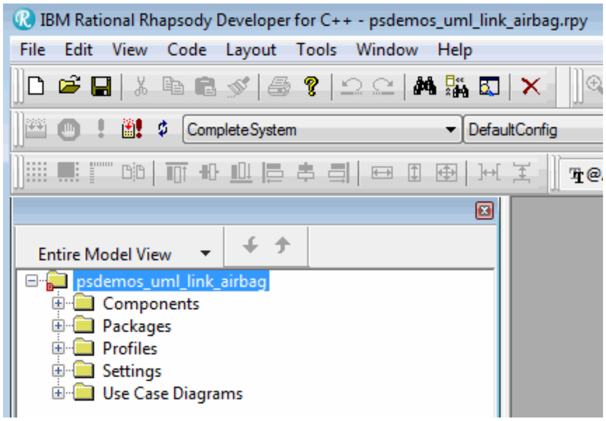
1. In the Rhapsody editor, select **File > Add Profile to Model**. The Add Profile to Model dialog box opens.
2. Navigate to the folder *`Polyspace_Install`*`\polyspace\plugin\rhapsody\profiles\Polyspace`.
3. Select the file `Polyspace.sbs`. Then click **Open**.

Now, if you right-click a package or file, you see the **Polyspace** item in the context menu. Selecting **Polyspace** opens the Polyspace Verification dialog box.

Accessing Polyspace Features

To access Polyspace features in the Rhapsody editor:

1. Open the model that you want to verify. For example, `psdemos_uml_link_airbag.rpy` in *`Polyspace_Install`*`/polyspace/plugin/rhapsody/psdemos`. *`Polyspace_Install`* is the location of the Polyspace installation folder.

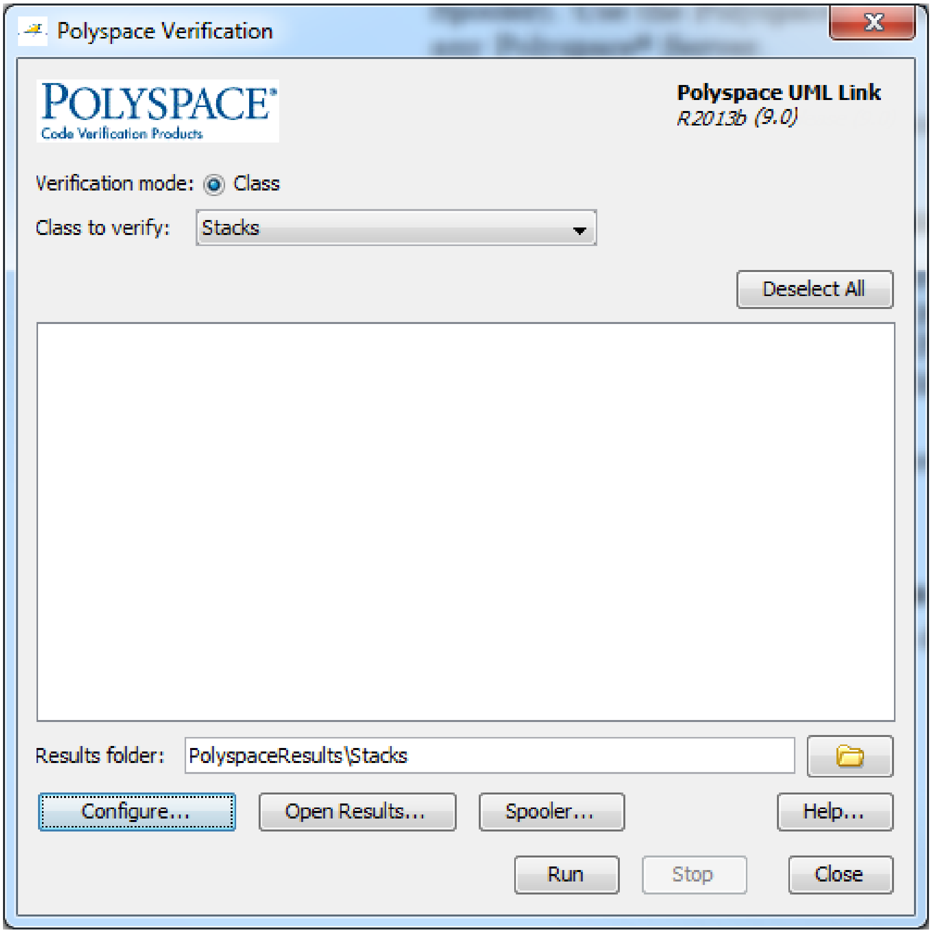


2 In the **Entire Model View**, expand the `Packages` node.

3 Right-click a package, for example, **AirBagFiles**.

4 From the context menu, select **Polyspace**.

The Polyspace Verification dialog box opens.



Polyspace Verification

Polyspace UML Link
R2013b (9.0)

Verification mode: ● Class

Class to verify: Stacks

Results folder: PolyspaceResults\Stacks

Through the Polyspace Verification dialog box, you can:

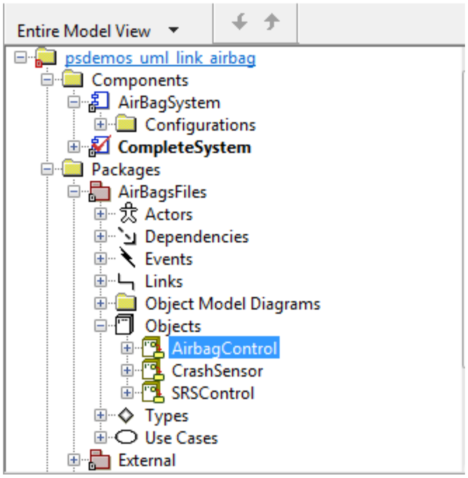
- Specify verification options. See "Configuring Verification Options" on page 5-6.
- Start verifications. See "Running a Verification" on page 5-6.
- Stop client-based verification. See "Running a Verification" on page 5-6.
- View verification results. See "Viewing Polyspace Results" on page 5-7.
- Open help.

- Open the Polyspace Queue Manager. See “Monitoring a Verification” on page 5-7.

Configuring Verification Options

To specify options for your verification:

1. In the **Entire Model View**, right-click a package or class, for example, `AirbagControl`.



2. From the context menu, select **Polyspace**.

3. In the Polyspace Verification dialog box, click **Configure**. The **Configuration** pane of the Polyspace verification environment opens.

4. Select options for your verification.

5. To save your options, on the toolbar, click 💾.

For information on how to choose your options, see “Analysis Options”.

Running a Verification

Before starting a verification, make sure that the generated code for the model is up to date.

To start a verification:

1. In the Rhapsody editor, select **Tools > Polyspace**. The Polyspace Verification dialog box opens.

2. In the **Results folder** field, specify a location for your verification results.

3. By default, the **Verification mode** is **Class**.

4. Click **Run**. In the **Log** view of the Rhapsody editor, you see verification messages.

To stop a client verification, in the Polyspace Verification dialog box, click **Stop**.

To stop a verification on the Polyspace Server, use the Polyspace Queue Manager. See "Monitoring a Verification" on page 5-7.

Monitoring a Verification

If your verification is client-based, you can observe progress in the **Log** view of the Rhapsody editor.

If your verification is running on a Polyspace Server:

1. In the Rhapsody editor, select **Tools > Polyspace**.

2. In the Polyspace Verification dialog box, click **Spooler**.

   Use the Polyspace Queue Manager to manage jobs running on a Polyspace Server.

   For more information, see "Verification Management".

Viewing Polyspace Results

To view results from the last completed verification:

1. In the Rhapsody editor, select **Tools > Polyspace**.

2. In the Polyspace Verification dialog box, click **Open Results**.

The Polyspace verification environment opens, displaying results in the Results Manager perspective.

For more information on Polyspace verification results, see "Run-Time Error Review".

Locating Faulty Code in Rhapsody Model

To identify the faulty code within your Rhapsody model using Polyspace verification results:

1. In the Results Manager perspective of the Polyspace verification environment, navigate to an error.

2. In the Source pane, right-click the error. From the context menu, select **Back To Model**.

Tip For the **Back To Model** command to work, you must have your Rhapsody model open.

The **Back To Model** command works best when the Polyspace check is enclosed by the tags `//#[` and `]#//`.

The software locates the faulty code within your Rhapsody model. Depending on the Rhapsody configuration, the faulty code appears either in a dialog box or in the code view.

The 64-bit version of the Polyspace product supports the **Back To Model** command only for version 8.0 of the IBM Rational Rhapsody product. For other versions, use the 32-bit Polyspace version.

To install the 32-bit Polyspace version, from a DOS command window, run the following command:

```
DVD\Installer32bits\Windows\Disk1\InstData\VM\Polyspace.exe
```

Template Configuration Files

- "Using Template Configuration Files" on page 5-9

- "Default Configuration Options" on page 5-9

Using Template Configuration Files

The first time you perform a verification, the software copies a template, Polyspace configuration file, from *Polyspace_Install*/polyspace/plugin/rhapsody/etc/template_*language*.psprj to the project folder. The software also renames the copy *model_language*.psprj, where:

- *model* is the name of your model.
- *language* is the name of the language that the model targets, that is ADA.

You can update the template .psprj file by one of the following means:

- Editing it through the Polyspace verification environment
- Double-clicking the file in a Windows Explorer window
- Replacing the template file with a copy of the .psprj file from a Rhapsody model folder

You can then share a configuration among project members and use the configuration with other projects.

Default Configuration Options

The template_*language*.psprj XML files specify the default option values for code verification.

The file template_Ada.psprj is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright 1999-2012 The MathWorks, Inc. -->
<!-- DO NOT EDIT THIS FILE: Some changes can lead to a
     crash of Polyspace products -->
<polyspace_project name="template_psprj" language="Ada 95" author="polyspace"
version="1.0" date="08/04/2011" path="file:/C:/Program Files/Polyspace
/polyspace/plugin/rhapsody/etc/template_ADA.psprj" rev="1.3">
  <source>
  </source>
  <include>
  </include>
  <module name="Verification_1" isactive="true">
    <source>
    </source>
    <optionset name="template_psprj" isactive="true">
      <option flagname="-OS-target">no-predefined-OS</option>
    </optionset>
  </module>
</polyspace_project>
```