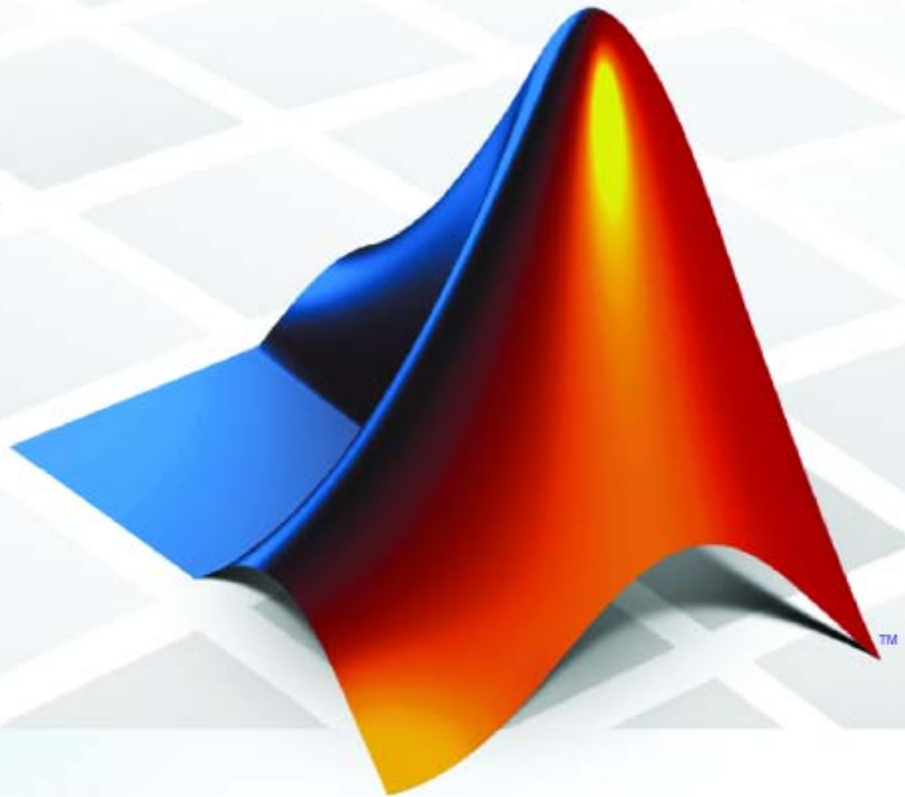


# PolySpace™ Client/Server for C++ 5

## Getting Started Guide



## How to Contact The MathWorks



www.mathworks.com  
comp.soft-sys.matlab  
www.mathworks.com/contact\_TS.html

Web  
Newsgroup  
Technical Support



suggest@mathworks.com  
bugs@mathworks.com  
doc@mathworks.com  
service@mathworks.com  
info@mathworks.com

Product enhancement suggestions  
Bug reports  
Documentation error reports  
Order status, license renewals, passcodes  
Sales, pricing, and general information



508-647-7000 (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™ Client/Server for C++ Getting Started Guide*

© COPYRIGHT 1997–2008 by The MathWorks, Inc.

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

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

### Trademarks

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

### Patents

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

### Revision History

March 2008      First printing      Revised for Version 5.1 (Release 2008a)

## General Requirements

### 1

<b>Computer Configuration</b> .....	<b>1-2</b>
<b>Timing Information</b> .....	<b>1-3</b>
<b>Installation Guide</b> .....	<b>1-4</b>
<b>Structure of This Document</b> .....	<b>1-5</b>

## PolySpace™ Client – Setting Up and Launching an Analysis on a Single Class

### 2

<b>Overview</b> .....	<b>2-2</b>
<b>Analysis Prerequisites</b> .....	<b>2-3</b>
<b>Setting Up a PolySpace™ Client Analysis</b> .....	<b>2-4</b>
<b>PolySpace™ Client: Running the Analysis</b> .....	<b>2-12</b>
Starting the Analysis .....	<b>2-12</b>
Parsing Errors During Preliminary Analysis Stages .....	<b>2-14</b>
Progression of the Analysis .....	<b>2-20</b>
End of the Analysis .....	<b>2-22</b>

**3**

<b>Overview</b> .....	<b>3-2</b>
The PolySpace™ Class Analyzer .....	3-2
Sources to be Analyzed .....	3-2
Architecture of the Generated main .....	3-3
<b>Log File</b> .....	<b>3-4</b>
Log File Overview .....	3-4
Characteristics of a Class and Messages of the Log File ..	3-5
<b>Behavior of Global variables and members</b> .....	<b>3-6</b>
Global Variables .....	3-6
Data Members of Other Classes .....	3-7
<b>Methods and Classes Specificities</b> .....	<b>3-9</b>
Template .....	3-9
Abstract Classes .....	3-9
Static Classes .....	3-10
Inherited Classes .....	3-10

**PolySpace™ Viewer – Exploring Results**

**4**

<b>Overview</b> .....	<b>4-2</b>
<b>Modes of Operation</b> .....	<b>4-3</b>
<b>Download Results into the Viewer</b> .....	<b>4-5</b>
<b>Reviewing PolySpace™ Results in “Expert” Mode</b>	
<b>(training.cpp)</b> .....	<b>4-7</b>
Overview of Expert Mode .....	4-7
Procedural Entities View (RTE View) .....	4-9
Colors in the Source Code View .....	4-15
More Examples of Run-Time Errors .....	4-16

Advanced Results Exploration .....	4-19
C++ specific checks .....	4-23
Miscellaneous .....	4-23
<b>Methodological Assistant</b> .....	<b>4-25</b>
Methodological Assistant Overview .....	4-25
Opening the Methodological Assistant .....	4-25
Assistant Dashboard .....	4-26
Choose a Methodological Assistant .....	4-30
<b>Report Generation</b> .....	<b>4-32</b>

## Launch PolySpace™ Remotely

# 5

<b>Overview</b> .....	<b>5-2</b>
<b>Launching an Analysis</b> .....	<b>5-3</b>
<b>Managing Your Remote Analysis: the PolySpace™ Spooler</b> .....	<b>5-5</b>
<b>Batch Commands</b> .....	<b>5-8</b>
Launching an Analysis in Batch .....	5-8
Managing an Analysis in Batch .....	5-8
<b>Sharing Analyses Between Accounts</b> .....	<b>5-11</b>
Analysis-key.text File .....	5-11
Magic Key or Shared Analysis Between Projects .....	5-12

## Summary

# 6



# General Requirements

---

Computer Configuration (p. 1-2)	Describes how to access hardware requirements
Timing Information (p. 1-3)	Describes how long it takes to perform this tutorial
Installation Guide (p. 1-4)	Describes the PolySpace™ products available on the installation CD
Structure of This Document (p. 1-5)	Describes the exercises included in this document

## **Computer Configuration**

Minimum hardware requirements to follow this tutorial on a Microsoft® Windows® PC are described in the installation guide available from the PolySpace™ installation CD-ROM (`\\Docs\Install\PolySpace_Installation_Guide.pdf`).



## Timing Information

The installation of PolySpace™ products takes around 5 minutes (the complete installation guide is available from the PolySpace installation CD-ROM in \Docs\Install\PolySpace\_Installation\_Guide.pdf).

- The first step of this tutorial takes about 15 minutes.
- The second step of this tutorial takes about 15 minutes.
- The third step of this tutorial takes about 5 minutes.

## Installation Guide

---

**Note** If the PolySpace™ products are already installed on your computer, please go directly to Chapter 2, “PolySpace™ Client – Setting Up and Launching an Analysis on a Single Class”.

---

The PolySpace products are delivered on a CD-ROM. There are 4 modules:

- PolySpace™ Client™ for C/C++ product – analyzing single class.
- PolySpace™ Server™ for C/C++ product – analyzing classes or composite analysis.
- *PolySpace Viewer* is the graphical user interface to explore the results computed by PolySpace Client or PolySpace Server.
- *PolySpace Spooler* is the graphical interface to manage analysis sent in remote.

Please refer to the PolySpace Installation Guide for installing the PolySpace products.

## Structure of This Document

Once the installation is done, you can launch PolySpace™ software by using the following icons that were placed on your desktop:



This Getting Started Guide will focus on the following steps using PolySpace products:

- In Chapter 2, “PolySpace™ Client – Setting Up and Launching an Analysis on a Single Class” we will analyze a simple class in `training.cpp` by using the class analyzer available in PolySpace™ Client™ for C/C++.
- In Chapter 3, “Class Analyzer” we will describe the capabilities of the class analyzer.
- In Chapter 4, “PolySpace™ Viewer – Exploring Results” we will review analysis results by using PolySpace Viewer.
- In Chapter 5, “Launch PolySpace™ Remotely”, instead of performing a PolySpace analysis locally, we will send it remotely to a server.



# PolySpace™ Client – Setting Up and Launching an Analysis on a Single Class

---

Overview (p. 2-2)

Provides an overview of the exercise performed in this chapter

Analysis Prerequisites (p. 2-3)

Describes requirements for performing the analysis

Setting Up a PolySpace™ Client Analysis (p. 2-4)

Describes how to set up the analysis

PolySpace™ Client: Running the Analysis (p. 2-12)

Describes how to run the analysis

### Overview

This chapter describes a basic class analysis. It focuses on the analysis of MathUtils class of `training.cpp`, which is included in the PolySpace™ installation directory and located at:

`PolySpaceInstallDir\Examples\Demo_Cpp_Long\sources\training.cpp`.

The PolySpace analysis process is composed of three main phases:

- 1** First, PolySpace checks the syntax and semantic of the analyzed file(s). However, as PolySpace is not associated to a particular compiler, **benefits** of this phase are triple for the analyzed source code: **ANSI C++ compliance, portability and maintainability**.
- 2** Then, PolySpace seeks the main procedure. If none is found, PolySpace Client for C/C++ will generate one automatically. By default, the main will build an instance of the class using the constructor and call all its public and protected function methods.
- 3** Finally, PolySpace proceeds with the code analysis phase, during which run time errors are detected and highlighted in the code.

## Analysis Prerequisites

Any analysis requires the following:

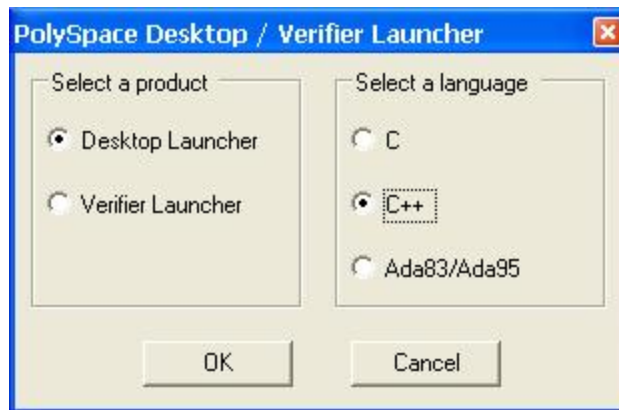
- PolySpace Client For C/C++ product and its related license file correctly installed;
- Source code files (in this case `training.cpp`) and all header files that it may directly or indirectly include. For this tutorial we will see later that we need three header files, `training.h`, `zz_utils.h`, and `math.h` in order to analyze the class `MathUtils` in `training.cpp`.
- All “-D” compilation switches necessary to compile the file are known. Please note that in this tutorial, no “-D” is necessary to compile `training.cpp`.

## Setting Up a PolySpace™ Client Analysis

- 1 Double-click on the PolySpace Launcher icon:



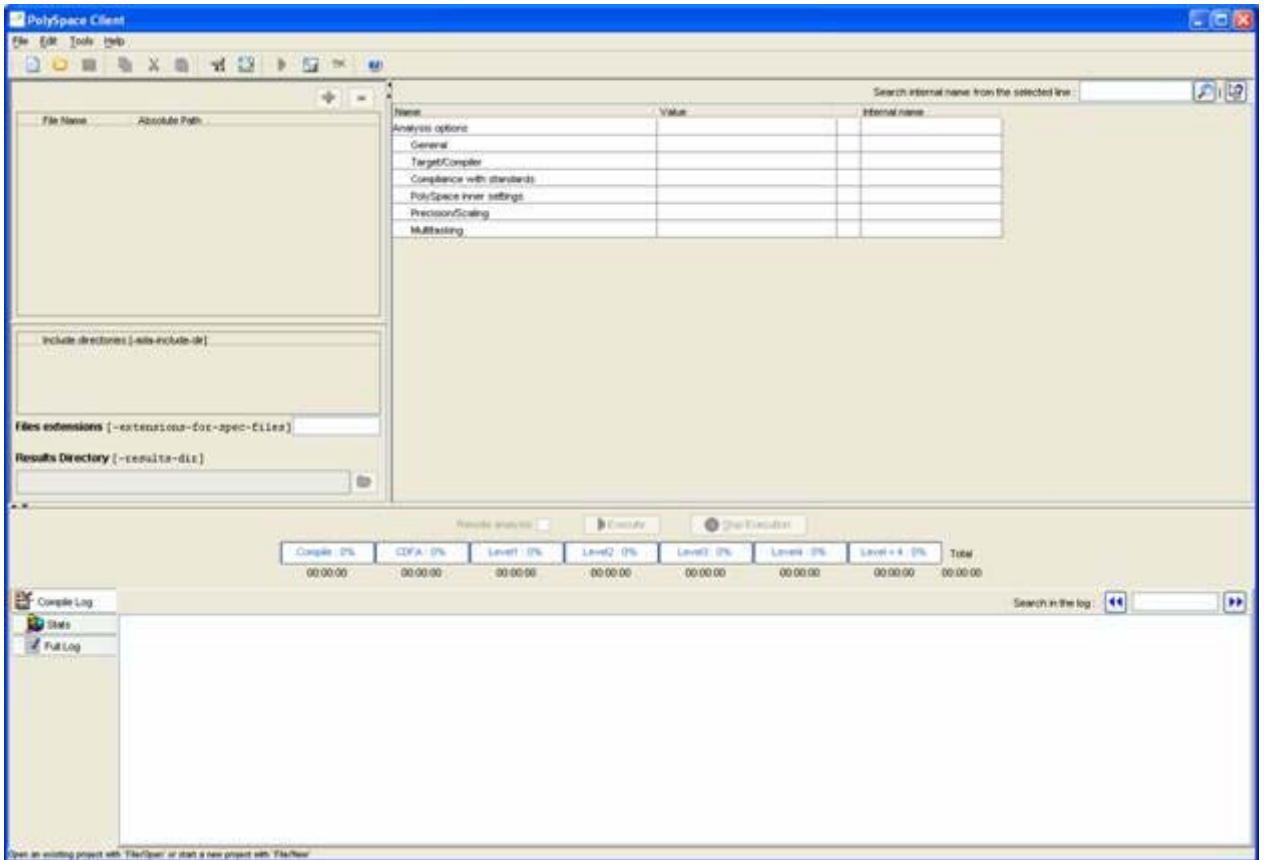
A dialog box window appears proposing to launch one of the following categories of analysis mixing the type of product and the language:



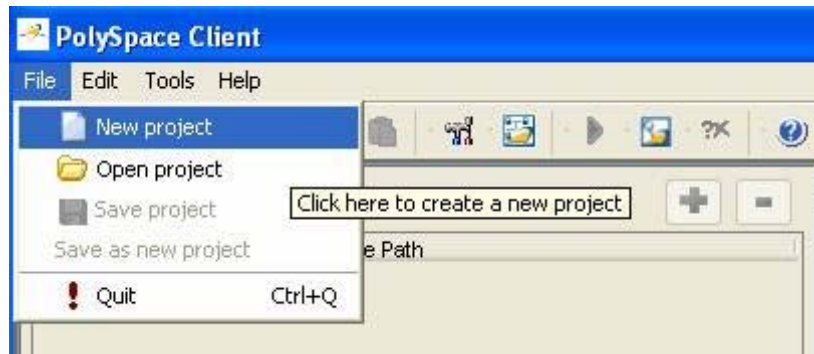
- 2 Select **Desktop Launcher**, and **C++**, then click **OK**.

The Graphical Interface of PolySpace analysis Launcher is displayed as below:



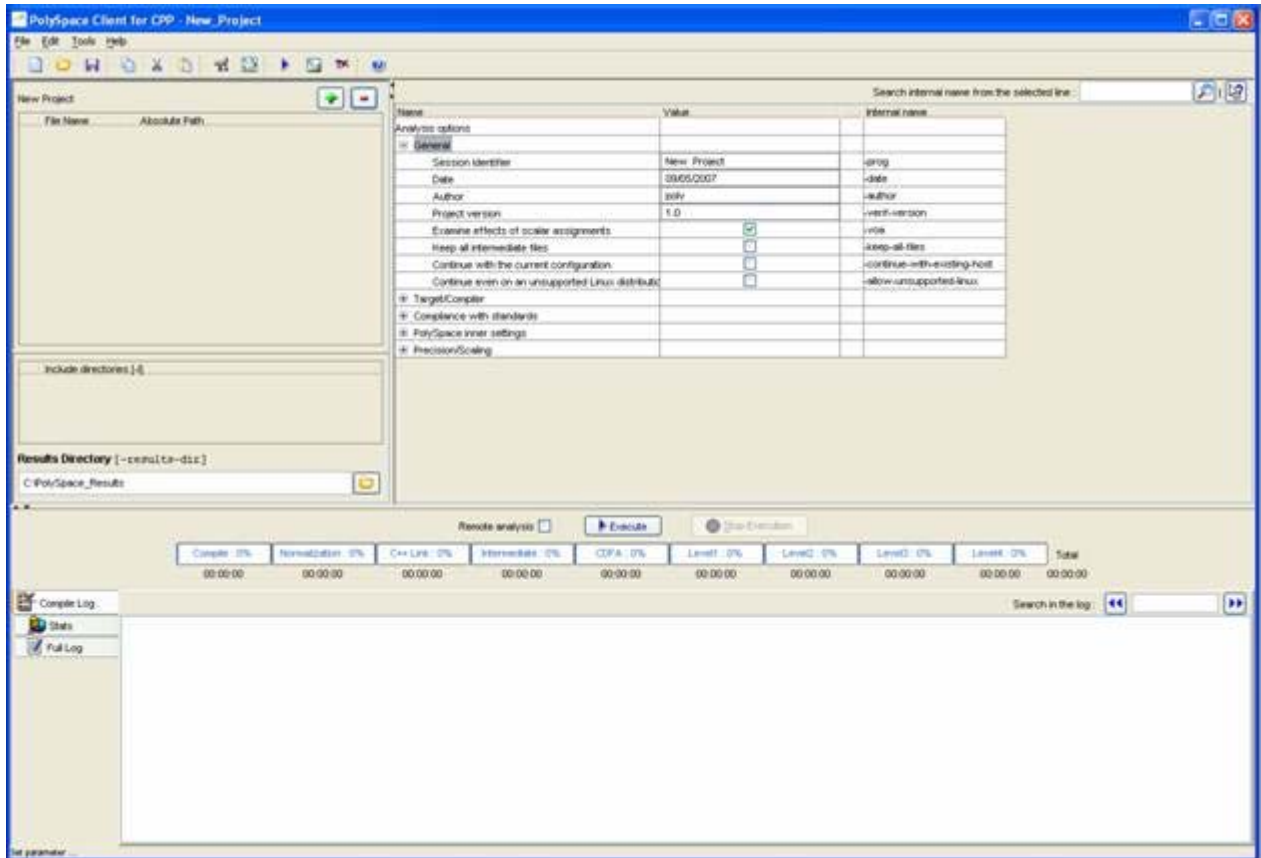


**3** Select **File > New Project** to start an analysis:




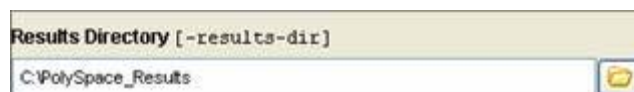
The PolySpace Client New Project window opens. It contains four sections:

- At the very top, the title bar, which contains usual icons and menus;
- Top left is the list of files to analyze, along with include and results directories;
- Top right is the set of options associated with the analysis that will be processed;
- Finally the bottom area allows following the execution and progress of the analysis.




**Note** it is also possible to drag a directory or source files and drop them directly in the “File Name/Absolute Path” part (top left of PolySpace Client) without using the “Please select a file” window.

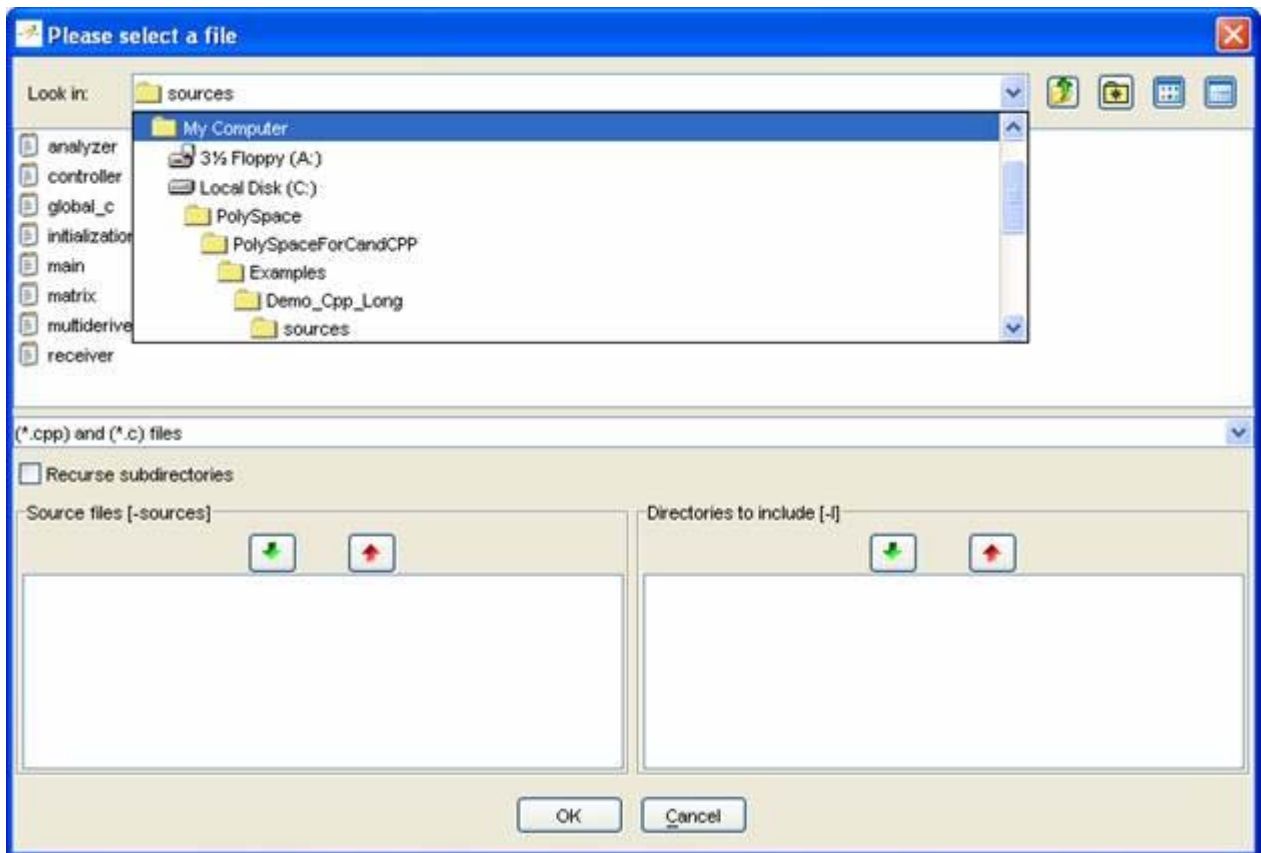
- 4 Start by updating the result directory name by clicking on the browse button .




This directory is the one where PolySpace Client will store the results of the analysis. By default, PolySpace will store results in `C:\PolySpace_Results`. This is the directory that we will choose for the analysis.


- 5 Now, Click on the  button (right of the “New Project” label).

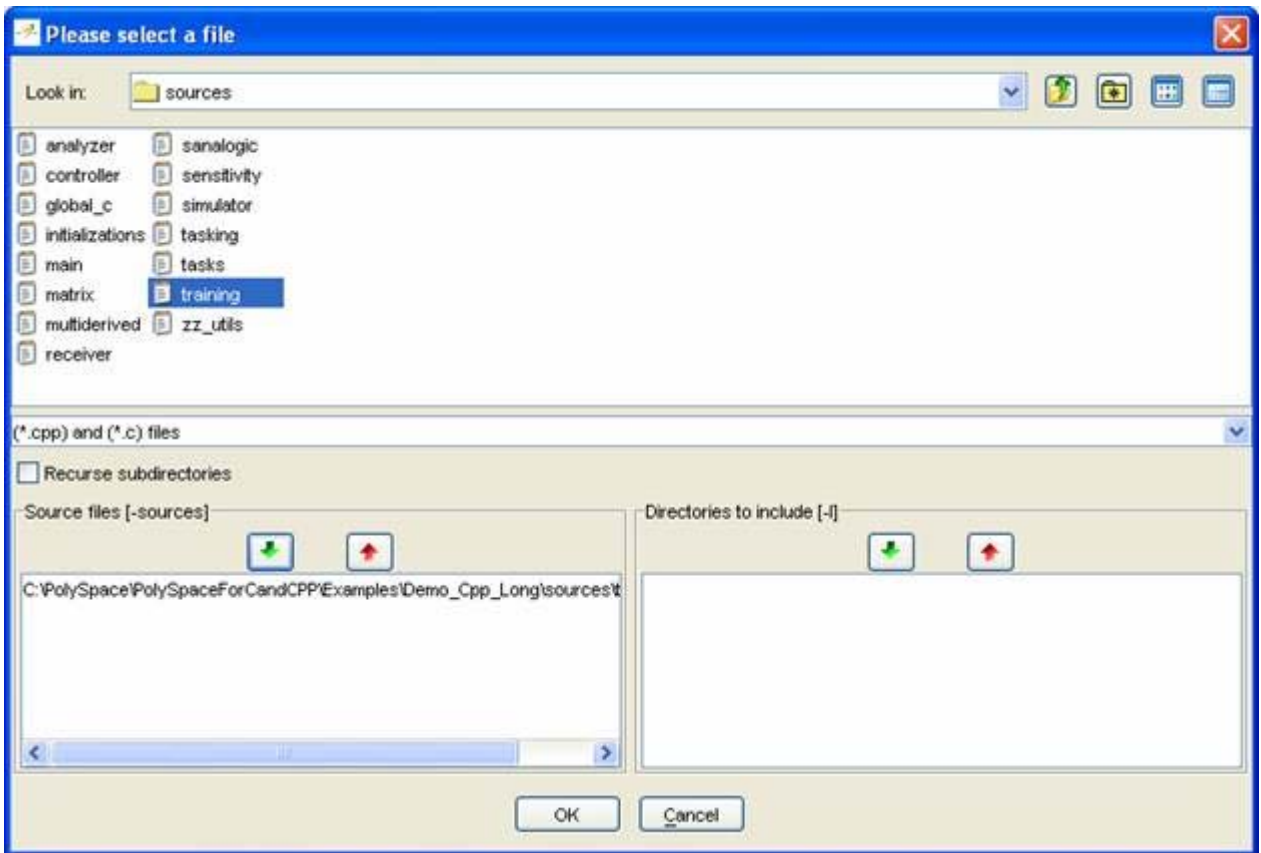
It opens the “Please select a file” window, from which you can select one or several files to analyze.



- 6 In the “Look in” section, click on , and select `PolySpaceInstallDir\Examples\Demo_Cpp_Long\sources`. A list

of files appears in the box (*PolySpaceInstallDir* corresponds to `C:\PolySpace\PolySpaceForCandCPP` in the figure above).

- 7 Select “training.cpp” and click on  in the “Source files [-sources]” section (bottom left) of the window. The file is now listed among the source files to be analyzed.



- 8 Click on OK to go back to the “PolySpace Client for CPP - New\_Project” window.

**Note** it is also possible to drag a directory or source files and drop them directly in the “File Name/Absolute Path” part (top left of PolySpace Client) without using the “Please select a file” window.

- 9 Now, click on  and expand the “PolySpace inner settings” group.
- 10 Check the box  in the “Generate a main” column that is associated to the “-main-generator” line as shown below. It enables the “-class-analyzer” option allowing to give the name of the class to analyze (see also step 2). For the needed of this tutorial, please type “MathUtils” in the column at the centre as shown in the figure below. When the class is surrounded by a name space, use the standard C++ syntax `<namespace>::<classname>`.

Name	Value		Internal name
Analysis options			
[-] General			
Session identifier	New Project		-prog
Date	10/05/2006		-date
Author	bard		-author
Project version	1.0		-verif-version
Examine effects of scalar assignments		<input checked="" type="checkbox"/>	-voa
Keep all intermediate files		<input type="checkbox"/>	-keep-all-files
Continue with the current configuration		<input type="checkbox"/>	-continue-with-existing-host
Continue even on an unsupported Linux distribution		<input type="checkbox"/>	-allow-unsupported-linux
[+] Target/Compiler			
[+] Compliance with standards			
[-] PolySpace inner settings			
[+] Specify a Visual kind of main			
[-] Generate a main using a given class			
Class name	MathUtils	<input checked="" type="checkbox"/>	-class-analyzer
Analyze only the given class		<input type="checkbox"/>	-class-only
Functions called by the generated main	default		-class-analyzer-calls
[+] Generate a main using given functions			
[+] Stubbing			
[+] Assumptions			
[+] Others			
[+] Precision/Scaling			
[+] Multitasking			

**11 It is also recommended** to select the `-voa` option. This option allows you to give some information on each scalar assignment with possible range of values. It can help understanding PolySpace messages.

---

**Note** When you want to analyze classes alone, you can check the `-class-only` option. It means that even if you add any other classes and function member definitions, PolySpace will stub them. This option accelerates analysis and allows you to check **robustness** issues only on the class. For this tutorial, it is not necessary to check this option; the “MathUtils” class does not depend on other classes.

---

## PolySpace™ Client: Running the Analysis

In this section...
“Starting the Analysis” on page 2-12
“Parsing Errors During Preliminary Analysis Stages” on page 2-14
“Progression of the Analysis” on page 2-20
“End of the Analysis” on page 2-22

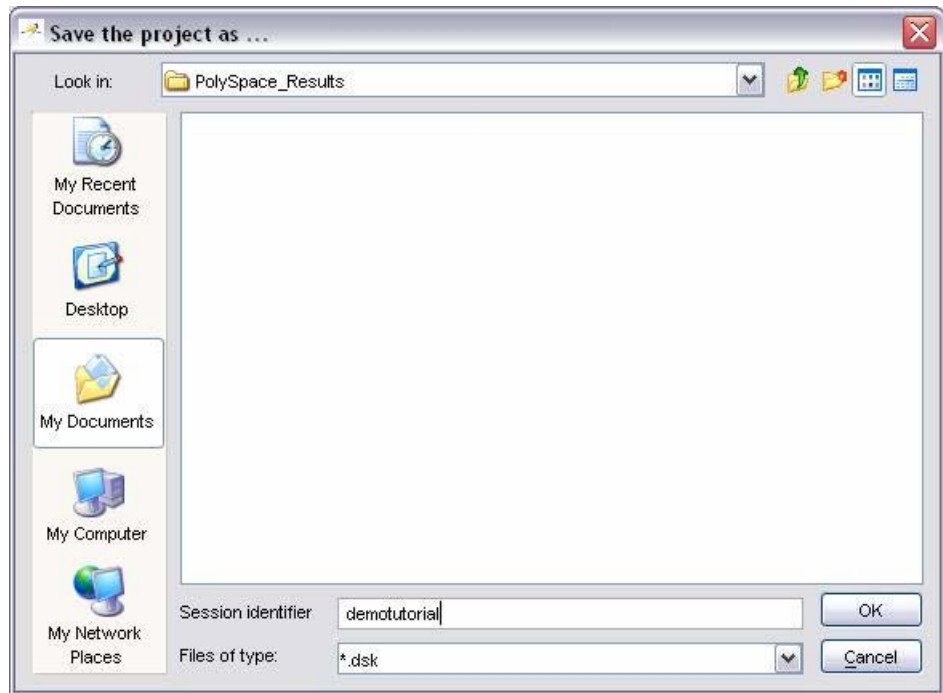
### Starting the Analysis

To run the analysis:

- 1 Click on  to start the analysis. Alternatively, you can click on the button in the title bar to run PolySpace Client with the current setting.

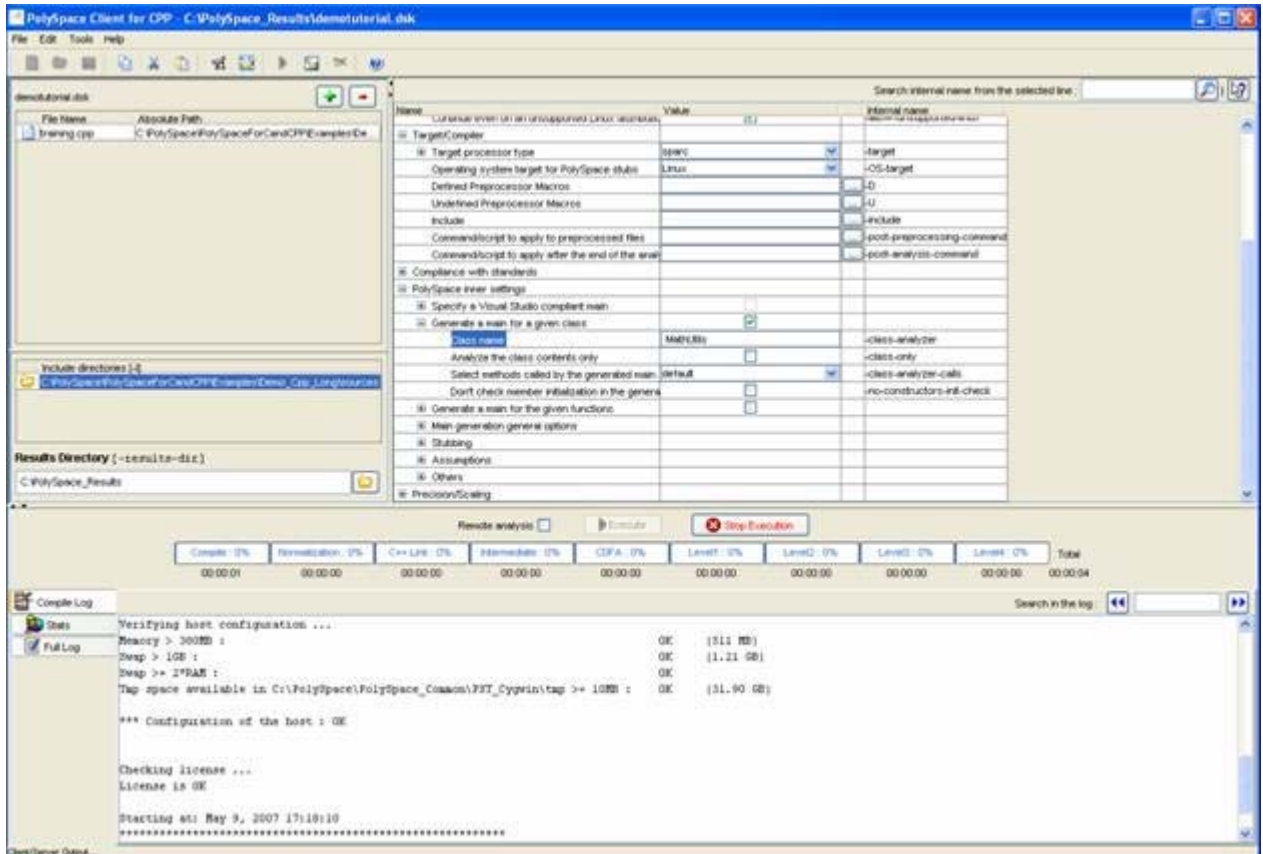
The window titled “Save the project as” opens. You can decide where to store the configuration information related to the analysis. Here, create a file called `demotutorial` and save it under PolySpace result directory. The full name of that file will be `demotutorial.dsk`.






- 2 Click **OK** to go back to the “PolySpace Client for CPP - New\_Project” window and click again on  to proceed.

## 2 PolySpace™ Client – Setting Up and Launching an Analysis on a Single Class

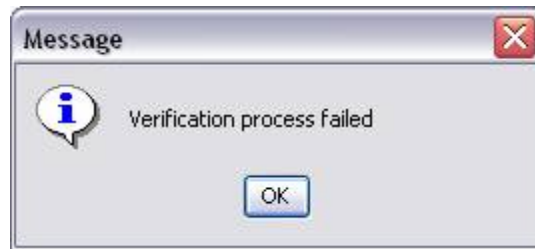


A progress report is displayed in the bottom part of the graphical interface, indicating that the analysis is being performed. The **Execute** button is also grayed out.

**Note** You may press the Stop Execution button -  to interrupt the analysis but it is not part of the current tutorial.

## Parsing Errors During Preliminary Analysis Stages

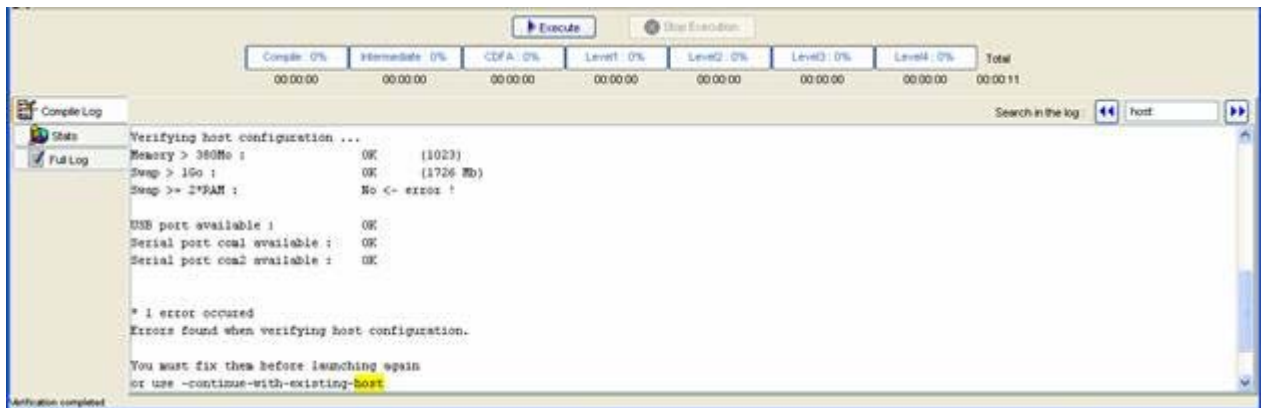
After some checks, PolySpace will show an error message:




Let's try and understand why we get this error message.

### First Possible Cause for the Error Message: Hardware Recommendation

If this happens, please verify whether your computer fits the minimal hardware configuration requirements described in the general requirements. Moreover, a message like the following one is displayed in the bottom part of the graphical interface:



- 1 Type “host” in the “Search in the log:” box and click on  to search if the error corresponds to a hardware recommendation problem.

If the error message corresponds to the one shown above and in order to continue analysis, you can either:

- upgrade your computer to meet the minimal requirements, or

- use the `-continue-with-existing-host` option which overrides the initial check for minimal hardware configuration. To do so, please follow the following steps:

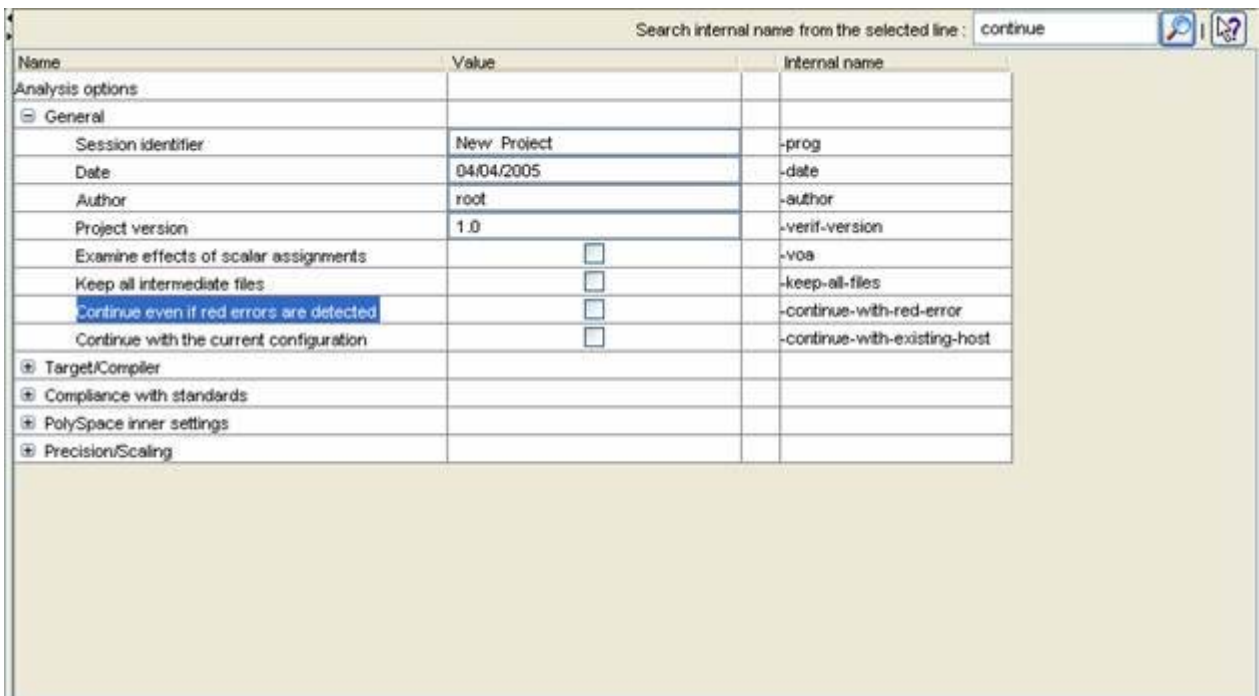
2 To set up the `-continue-with-existing-host` option, please type “continue” in the Search internal name from the selected line (top right box).



Search internal name from the selected line : continue

3 Click .

It will show all options containing “continue” in the set of options part below:

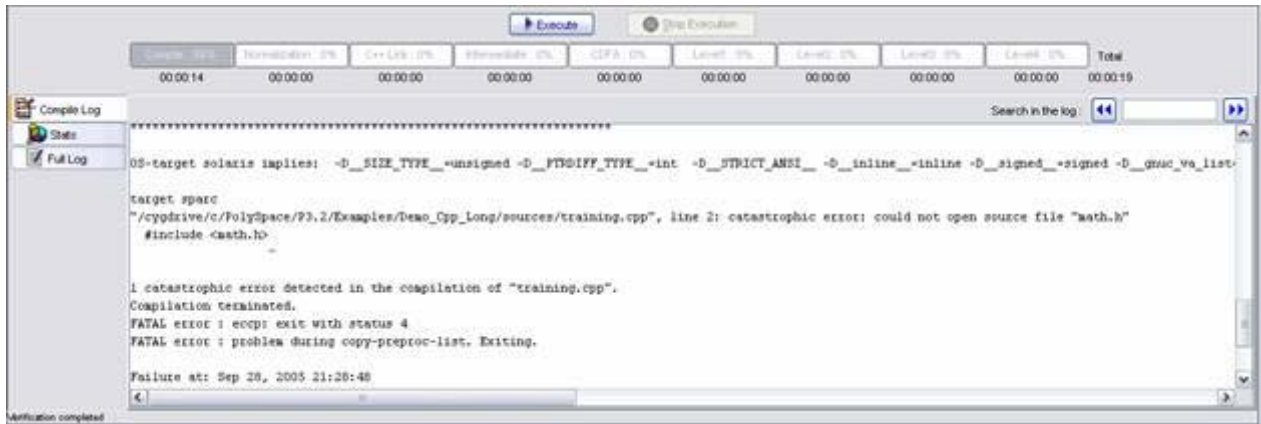


Name	Value	Internal name
Analysis options		
[-] General		
Session identifier	New Project	-prog
Date	04/04/2005	-date
Author	root	-author
Project version	1.0	-verif-version
Examine effects of scalar assignments	<input type="checkbox"/>	-voa
Keep all intermediate files	<input type="checkbox"/>	-keep-all-files
Continue even if red errors are detected	<input checked="" type="checkbox"/>	-continue-with-red-error
Continue with the current configuration	<input checked="" type="checkbox"/>	-continue-with-existing-host
[+] Target/Compiler		
[+] Compliance with standards		
[+] PolySpace inner settings		
[+] Precision/Scaling		

4 Check the box  in the “Value” column that is associated to the “`-continue-with-existing-host`” line as shown below.


## Second Possible Cause for the Error Message: Information About Header Files

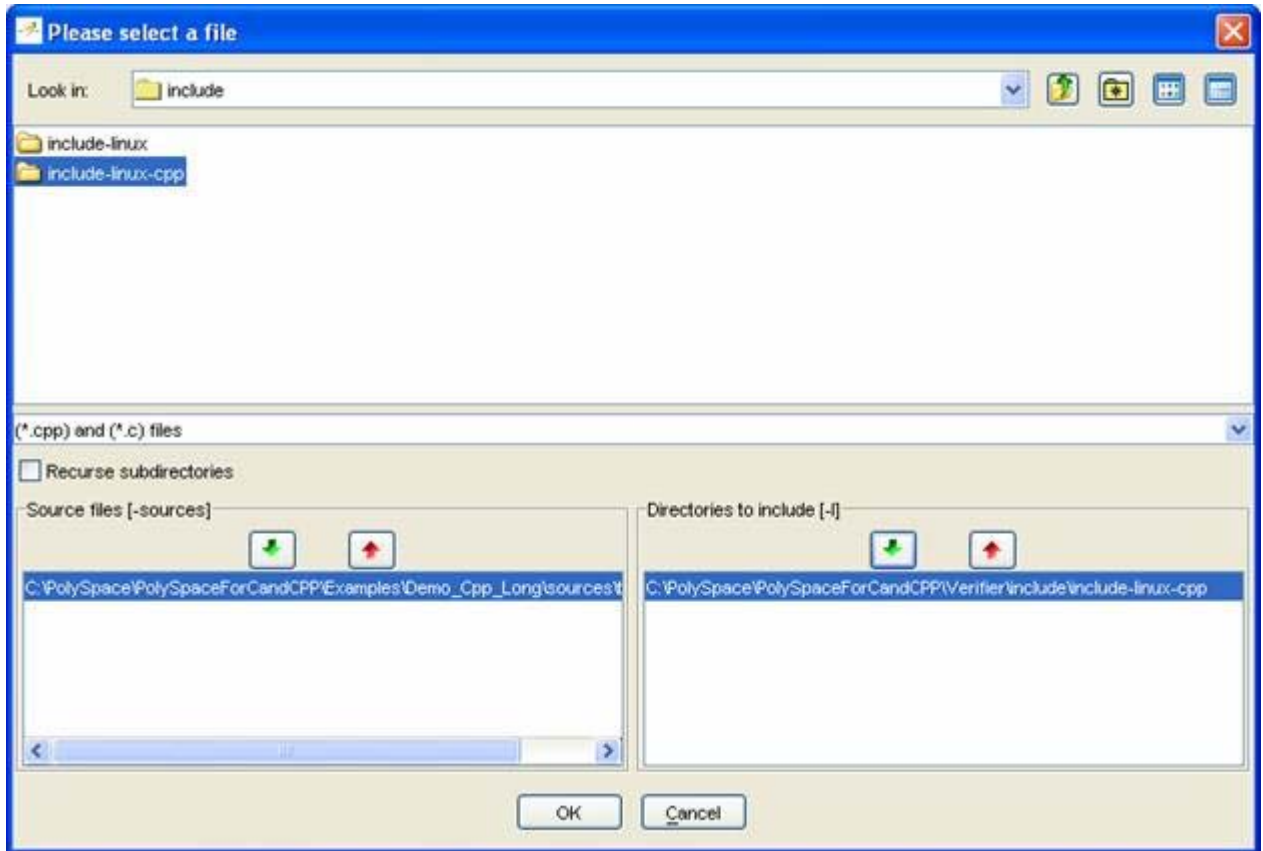
Another cause of error may be that PolySpace Desktop misses some package specifications.




In the tutorial, as shown above, the file named `math.h` cannot be found. To fix this problem, you need to indicate its location. As PolySpace is not associated with one particular compiler, it is mandatory to indicate where library files are stored.

In our `training.cpp` file analysis, the related `math.h` file is distributed with the PolySpace C++ product in `PolySpaceInstallDir\include\include-linux-cpp`. This distribution concerns a Linux® OS target and is only given as material of help. For analyzing your code, it is recommended to indicate the path to the standard headers dedicated to your own compiler.


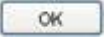
- 1 Open the “Please select a file” window using the  button (right of the “demotutorial.dsk” label in the top right of the interface):



2 Select *PolySpaceInstallDir\Verifier\include\include-linux-cpp*, where an exemplary of <“math.h”> is located for the Linux OS target.

3 Click on  in the “Directories to include [-I]” section.

Select *PolySpaceInstallDir\Examples\Demo\_Cpp\_Long\sources*, where *training.h* is located.

4 Click on  in the “Directories to include [-I]” section, then close the window using .

---

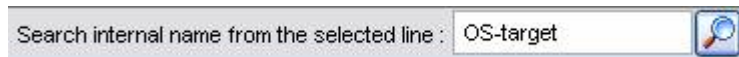
**Note** Other header file needed `zz_utils.h` is also located in same directory.


It is also possible to drag a directory and drop it directly in the “include directories [-I]” part (top left of PolySpace Desktop) without using the “Please select a file” window.

---


In this tutorial, as we have chosen includes of the OS Linux distribution, we have to select a Linux OS target. It defines a set of predefined compilation flags, known to be default or implicit compile options from cross-compiler for these platforms:

- 5 To set up the `-OS-target` Linux option, enter “OS-target” in the Search internal name from the selected line — top right box.



- 6 Then click on . It will show all options containing “OS-target” in the set of options part below:

Name	Value	Internal name
Analysis options		
[-] General		
Session identifier	New Project	-prog
Date	10/05/2006	-date
Author	bard	-author
Project version	1.0	-verif-version
Examine effects of scalar assignments	<input checked="" type="checkbox"/>	-voa
Keep all intermediate files	<input type="checkbox"/>	-keep-all-files
Continue with the current configuration	<input type="checkbox"/>	-continue-with-existing-host
Continue even on an unsupported Linux distribution	<input type="checkbox"/>	-allow-unsupported-linux
[-] Target/Compiler		
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
[+] Compliance with standards		
[-] PolySpace inner settings		
[+] Specify a Visual kind of main	<input type="checkbox"/>	
[+] Generate a main using a given class	<input checked="" type="checkbox"/>	
Class name	MathUtils	-class-analyzer
Analyze only the given class	<input type="checkbox"/>	-class-only
Functions called by the generated main	default	-class-analyzer-calls
[+] Generate a main using given functions	<input type="checkbox"/>	
[+] Stubbing		
[+] Assumptions		
[+] Others		
[+] Precision/Scaling		
[+] Multitasking		

7 Then, click on the  , allowing to chose linux OS target out of some predefined Operating System targets in the list solaris, linux, vxworks, no-predefined-OS and visual.

---

**Note** Associated to chosen Operating System (except no-predefined-OS), PolySpace dedicates a set of accurate stubs concerning standard templates and C libraries.

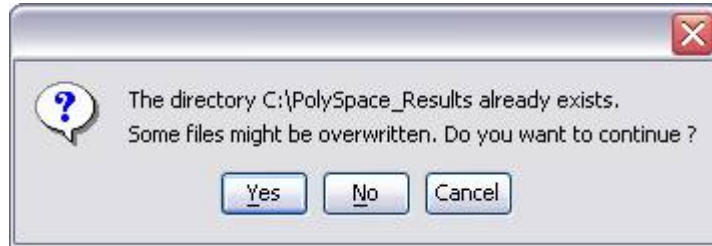
---

## Progression of the Analysis

1 Click on  to restart the analysis.




If you previously clicked **Execute**, some results may have already been written in the C:\PolySpace\_Results directory. Therefore a window opens to check whether you want to overwrite in this directory or not:



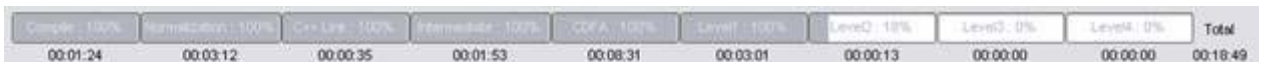
- 2 If this happens, click **Yes**.





---

**Note** Closing the PolySpace Desktop window will not stop the PolySpace analysis. If you wish to stop it, click  (a window of confirmation follows the click). If the window is closed without stopping the analysis, it continues in background. Opening again PolySpace Desktop with the same project automatically updates the analysis with its current status.

---

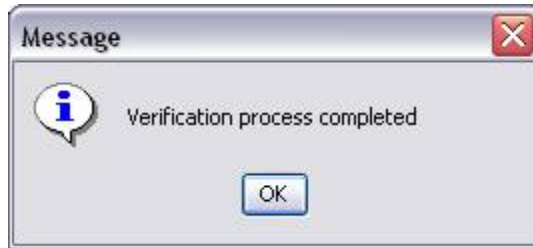
The progress bar allows to follow the progress of the analysis:




- 3 To obtain a progress report, click on  for the compilation phase, or  for the full analysis in the low level window.
- 4 Click  to get other pieces of information about current analysis (list of options, stubbed functions, functions used during main construction, checks found after each phase, etc.).
- 5 Click the  icon to refresh the summary.

## End of the Analysis

When the analysis ends, PolySpace proposes to review the results:



If you Click **OK**, go to the next section of the tutorial to view the results.

If you click **Cancel**, and no other analyses are running, you can access the results via the  icon in the title bar.

# Class Analyzer

---

Overview (p. 3-2)

Describes the class analyzer

Log File (p. 3-4)

Describes the contents of the analysis log file

Behavior of Global variables and members (p. 3-6)

Describes how the class analyzer treats global variables and data members of other classes

Methods and Classes Specificities (p. 3-9)

Describes specific restrictions on classes and methods

## Overview

In this section...
“The PolySpace™ Class Analyzer” on page 3-2
“Sources to be Analyzed” on page 3-2
“Architecture of the Generated main” on page 3-3

### The PolySpace™ Class Analyzer

PolySpace Class Analyzer analyses applications class by class, even if these classes are only partially developed.

**Benefits:** detecting errors at a very early stage, even if the class is not fully developed, without any test cases to write. The process is very simple: give the class name, and the PolySpace Class Analyzer will analyze its robustness:

- PolySpace will generate a “pseudo” main;
- It will call each constructor of the class;
- Then it will call each public function from the constructors;
- Each parameter will be initialized with full range (i.e. with a random value);
- External variables are also defined to random value.

---

**Note** Only prototypes of objects (classes, methods, variables, etc.) are needed to analyze a given class. All missing code will be automatically stubbed.

---

As a result, a class will be analyzed by exploring every branch of the methods through all its constructors (see some restrictions in the associated paragraph).

### Sources to be Analyzed

The functions that are analyzed include all non-inherited constructors and destructors, and all non-inherited public or protected methods of the class.

However, sources can also come from inherited classes (fathers) or be the sources of other classes used by the class that is being analyzed (friend, etc.).

For more information, see “PolySpace™ Class Analyzer Process” in the *PolySpace Client/Server for C++ User’s Guide*.

## Architecture of the Generated main

PolySpace generates the call to each constructor and method of the class. Each method will be analyzed with all constructors. Each parameter is initialized to random. However, even if you can have an idea of the architecture of the generated main in PolySpace Viewer, the main is not real. You cannot reuse and compile it with your analysis or PolySpace.

If we come back to the class “MathUtils”, analyzed in the first step, it contains one constructor, a destructor and seven public methods. The architecture of the generated main is as follows:

```
Generating call to constructor: MathUtils:: MathUtils ()
While (random) {
  If (random) Generating call to function: MathUtils::Pointer_Arithmetic()
  If (random) Generating call to function: MathUtils::Close_To_Zero()
  If (random) Generating call to function: MathUtils::MathUtils()
  If (random) Generating call to function: MathUtils::Recursion_2(int *)
  If (random) Generating call to function: MathUtils::Recursion(int *)
  If (random) Generating call to function: MathUtils::Non_Infinite_Loop()
  If (random) Generating call to function: MathUtils::Recursion_caller()
}
Generating call to destructor: MathUtils::~MathUtils()
```

---

**Note** An ASCII file representing the “pseudo” main can be seen in `C:\PolySpace_Results\ALL\SRC\__polyspace_main.cpp`.

If the class contains more than one constructor, they are called before the while statement in an if then else statement. From a PolySpace point of view, this architecture ensures that the analysis will evaluate each function method with every constructor.

---

## Log File

### In this section...

“Log File Overview” on page 3-4

“Characteristics of a Class and Messages of the Log File” on page 3-5

### Log File Overview

When analyzing a class, the list of methods used for the main is also given in the log file during the normalization phase of the C++ analysis.

You can have the details of what will be analyzed in the log. Here is the example concerning the MathUtils class and associated log file which can be found at root of C:\PolySpace\_Results:

```
*****
***
*** Beginning C++ source normalization
***
*****
Number of files : 1
Number of lines : 202
Number of lines with libraries : 7009
**** C++ source normalization 1 (Loading)
**** C++ source normalization 1 (Loading) took 20.8real, 7.9u + 11.4s
(1gc)
**** C++ source normalization 2 (P_INIT)
* Generating the Main ...
Generating call to function: MathUtils::Pointer_Arithmetic()
Generating call to function: MathUtils::Close_To_Zero()
Generating call to function: MathUtils::MathUtils()
Generating call to function: MathUtils::Recursion_2(int *)
Generating call to function: MathUtils::Recursion(int *)
Generating call to function: MathUtils::Non_Infinite_Loop()
Generating call to function: MathUtils::~~MathUtils()
Generating call to function: MathUtils::Recursion_caller()
```

It may happen that a main is already defined in the files you are analyzing. In this case, no other main will be generated, and this one will be analyzed. You will receive this warning:

```
*** Beginning C++ source normalization

* Warning: a main procedure already exists.
* No main will be generated: the existing one will be used
```

---

**Note** The main will be analyzed even if it does not concern the class given to the `-class-analyzer` option.

---

## Characteristics of a Class and Messages of the Log File

The log file may contain some error messages concerning the class to analyze. These messages appear when characteristics of class are not respected:

- It is not possible to analyze a class which does not exist in the given sources. The analysis will stop with the following message:

```
-----
@User Program Error: Argument of option -class-analyzer
must be defined : <name>.
Please correct the program and restart the verifier.
-----
```

- It is not possible to analyze a class which only contains declarations without code. The analysis will stop with the following message:

```
-----
@User Program Error: Argument of option -class-analyzer
must contain at least one function : <name>.
Please correct the program and restart the verifier.
-----
```

## Behavior of Global variables and members

In this section...
“Global Variables” on page 3-6
“Data Members of Other Classes” on page 3-7

### Global Variables

In a class analysis, global variables are not considered as following ANSI® Standard anymore. if they are defined and but not initialized. Remember that ANSI Standard considers, by default, that global variables are initialized to zero.

In a class analysis, global variables do not follow standard behavior:

- Defined variables: they are initialized to random. Then they follow the data flow of the code to analyze.
- Initialized variables: they are used with the initialized value. Then they follow the data flow of the code to analyze.
- External variables: the analysis will stop. To continue the analysis, it is mandatory to use the -allow-undef-variable option. In doing so, external variables follow the behavior of a defined variable.

An example below shows behavior of two global variables:

```
1
2 extern int fround(float fx);
3
4 // global variables
5 int globvar1;
6 int globvar2 = 100;
7
8 class Location
9 {
10 private:
11 void calculate_new(void);
12 int x;
```



```
13
14 public:
15 // constructor 1
16 Location(int intx = 0) { x = intx; };
17 // constructor 2
18 Location(float fx) { x = fround(fx); };
19
20 void setx(int intx) { x = intx; calculate_new(); };
21 void fsetx(float fx) {
22     int tx = fround(fx);
23     if (tx / globvar1 != 0) // ZDV check is orange
24     {
25         tx = tx / globvar2; // ZDV check is green
26         setx(tx);
27     }
28 };
29 };
```

In this example, globavar1 is defined but not initialized (see line 5): the check ZDV is orange at line 23. On the other hand, globvar2 is initialized to 100 (see line 6). The ZDV check is green at line 25.

## Data Members of Other Classes

When analyzing a specific class, variable members of other classes, even members of parent classes, are considered as initialized. They follow the following behavior:

- 1** They are considered as may be not initialized (unproven check NIV), if constructor of the class is not defined. So they are assigned to full range and then, they follow the data flow of the code to analyze.
- 2** They are considered as initialized to the value defined in the constructor, if the constructor of the class is defined in the class and given to the analysis. If -class-only option is used, it just like definition of constructor is missing (see item 1). Then they follow the data flow of the code to analyze.
- 3** They could be checked as run-time error, if and only if, the constructor is defined and does not initialize the considered member.

An example below shows the result of the analysis of the class `MyClass`. It shows behavior of a variable member of the class `OtherClass` given without definition of its constructor:

```
class OtherClass
{
protected:
    int x;
    OtherClass (int intx);    // code is missing
public:
    int getMember(void) {return x;}; // NIV is warning
};
class MyClass
{
    OtherClass m_loc;
public:
    MyClass(int intx) : m_loc(0) {};
    void show(void) {
        int wx, wl;
        wx = m_loc.getMember();
        wl = wx*wx + 2;    // Possible overflows because OtherClass
                          // member is assigned to full range
    };
};
```

In the example above, variable member of `OtherClass` is checked initialized to random: the check is orange at line 7 and there are possible overflows at line 17 because range of the return value `wx` is full range in the type definition.

## Methods and Classes Specificities

### In this section...

“Template” on page 3-9

“Abstract Classes” on page 3-9

“Static Classes” on page 3-10

“Inherited Classes” on page 3-10

### Template

A template class cannot be analyzed alone. Only instance of a template will be considered as the class that can be analyzed with the PolySpace Class Analyzer.

```
template<class T, class Z> class A { }
```

In the example above, we want to analyze template class A with two class parameters T and Z. For that, we have to define a “typedef” to create a specialization of the template, with a specific specialization for T and Z. In the example below, T represents a int and Z a double:

```
template class A<int, double>; // Explicit specialisation  
typedef class A<int, double> my_template;
```

my\_template is used as parameter of -class-analyzer option, to analyze the this instance of template A.

### Abstract Classes

In the real world an instance of an abstract class cannot be created, so it cannot be analyzed. However, it is easy to analyze by “removing” the pure declarations. For example, in an abstract class definition change:

```
void abstract_func () = 0; by void abstract_func ();
```

If an abstract class is given to analyze, the class analyzer will make the change automatically and the virtual pure function (in the example above abstract\_func) will then be ignored in the analysis of the abstract class.

This means that no call will be made from the generated main, so function is purely ignored. Moreover, if the function is called by another one, the pure virtual function will be stubbed and an orange check will be put on the call: “call of virtual function [f] may be pure”.

## Static Classes

If a class defines static methods, they are called in the generated main as a classical one.

## Inherited Classes

When a function is not defined in a derived class, even if it is visible because inherited from a fathers class, it is not called in the generated class. In the example below, the class Point derives from the class Location:

```
class Location
{
protected:
    int x;
    int y;
    Location (int intx, int inty);
public:
    int getx(void) {return x;};
    int gety(void) {return y;};
};
class Point : public Location
{
protected:
    bool visible;
public :
    Point(int intx, int inty) : Location (intx, inty)
    {
        visible = false;
    };
    void show(void) { visible = true;};
    void hide(void) { visible = false;};
    bool isvisible(void) {return visible;};
};
```

Since the two methods `Location::getx` and `Location::gety` are “visible” for derivated classes, the generated main does not include these methods when analyzing the class `Point`. No matter because, we have to analyze the `Location` class

However, inherited members are considered as volatile if there are not explicitly initialized in the fathers constructors. In the example above, the use of the two members `Location::x` and `Location::y` will be considered as volatile. Indeed, if we analyze the above example in the current state, the method `Location::Location(constructor)` will be stubbed.



# PolySpace™ Viewer – Exploring Results

---

Overview (p. 4-2)	Provides an overview of the exercise performed in this chapter
Modes of Operation (p. 4-3)	Describes the two ways you can review analysis results
Download Results into the Viewer (p. 4-5)	Describes how to open results into the viewer
Reviewing PolySpace™ Results in “Expert” Mode (training.cpp) (p. 4-7)	Describes how to use Expert mode
Methodological Assistant (p. 4-25)	Describes how to use the methodological assistant
Report Generation (p. 4-32)	Describes how to generate reports containing analysis results

### Overview

This step illustrates how to explore analysis results that were generated by either PolySpace Client or PolySpace Server. We review the results of the analysis of `training.cpp` performed in Chapter 2, “PolySpace™ Client – Setting Up and Launching an Analysis on a Single Class”.

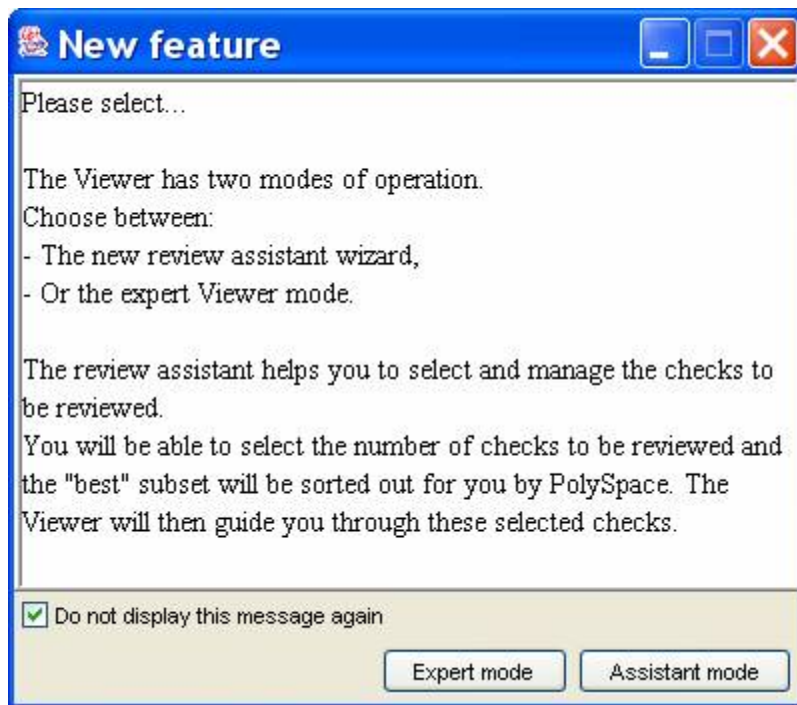


If you clicked **OK** at the end of the analysis in Chapter 2, PolySpace Viewer automatically opens results.



## Modes of Operation

The first time the PolySpace Viewer is opened, a sub-window will appear after the splash screen of the viewer. It is aimed to warn user about different modes of operation. User has to choose between launching the Viewer in an “expert” mode or in an “assistant” mode.



The mode will define the reviewing process of checks highlighted during an analysis:

- **Expert mode** —The Viewer is opened in a mode where all checks can be seen. The number, the order and the categories of checks to be reviewed have to be selected by the user himself (See next section).
- **Assistant mode** — The reviewing rules for a C++ analysis results follows a methodology selected by PolySpace. It concerns the “best” subset of

checks sorted out for user. The PolySpace Viewer will then guide user through these selected checks.

For the need of this tutorial, please untick “Do not display this message again” and then click on “Expert mode”.

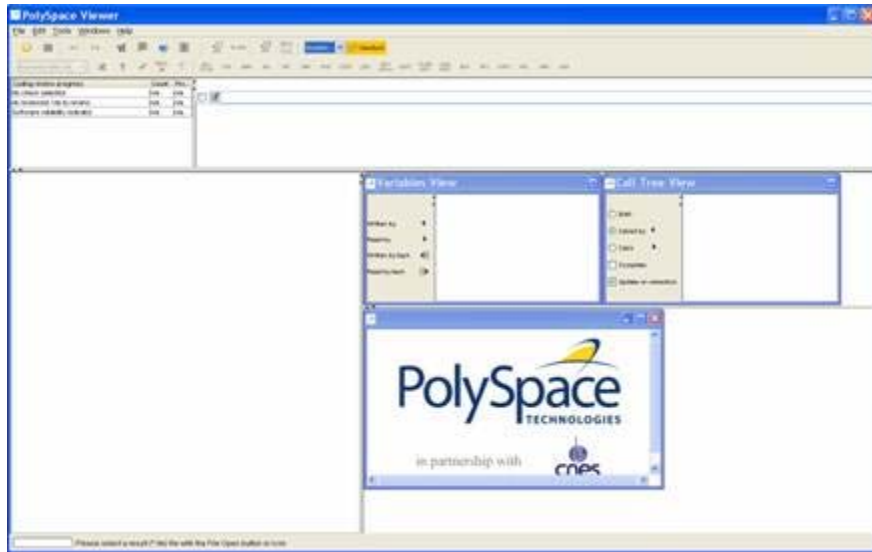
---

**Note** Even if the user has chosen one mode it is easy in one click to change the mode inside the PolySpace Viewer.

---

## Download Results into the Viewer

After having clicked on “Expert mode” the PolySpace Viewer window looks like the figure below:



- 1 Click **File > Open** to load result files.

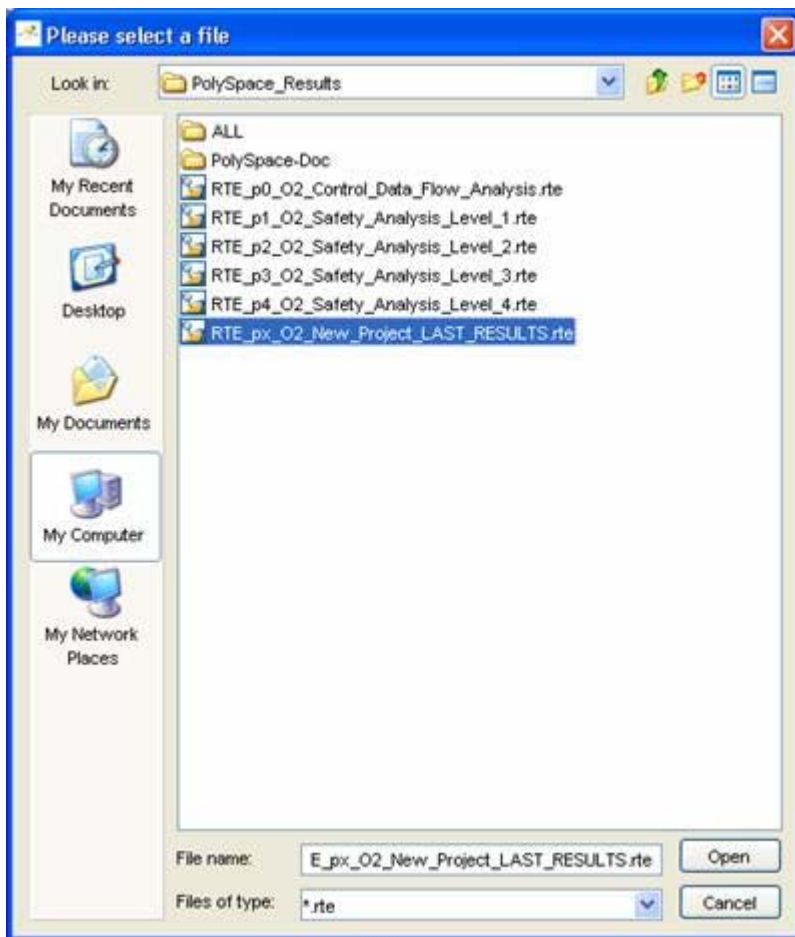
---

**Note** If you did not perform the analysis, you can still review the results by opening the following file:

*InstallDirectory*\Examples\Demo\_CPP\_Long\  
RTE\_px\_02\_Demo\_Cpp\_Long\_LAST\_RESULTS.rte. We will focus on  
training.cpp procedural entity.

---

- 2 Select the following file located in C:\PolySpace\_Results.



**3** Click **Open** to proceed with further steps

---

**Note** The `RTE_px_O2_Project Name_LAST_RESULTS.rte` file is a sort of “link” on the best analysis in term of precision. This analysis is represented by `RTE_p4_O2_Safety_Analysis_Level14.rte` file. Lower level files represent lower precision analysis.

---

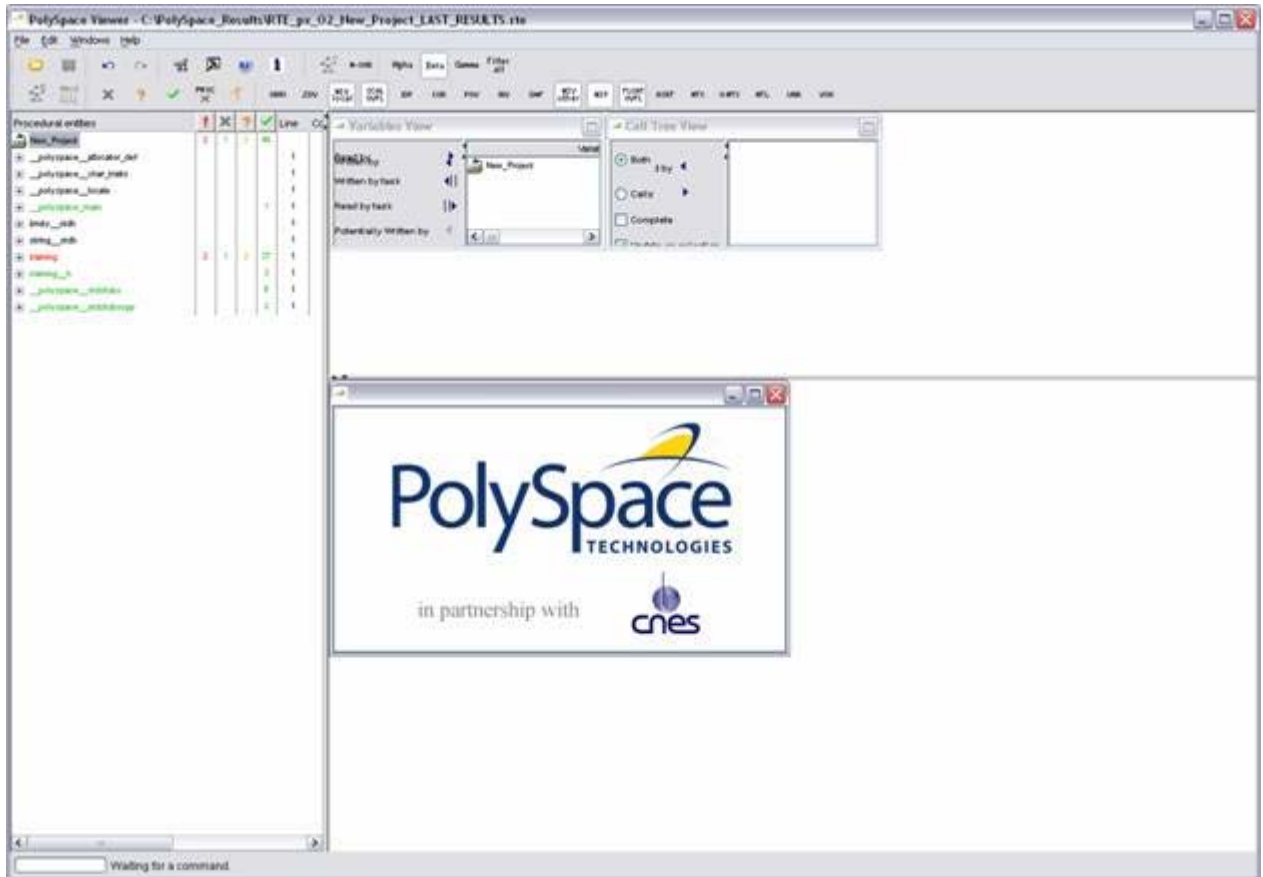
## Reviewing PolySpace™ Results in “Expert” Mode (training.cpp)

### In this section...

- “Overview of Expert Mode” on page 4-7
- “Procedural Entities View (RTE View)” on page 4-9
- “Colors in the Source Code View” on page 4-15
- “More Examples of Run-Time Errors” on page 4-16
- “Advanced Results Exploration” on page 4-19
- “C++ specific checks” on page 4-23
- “Miscellaneous” on page 4-23

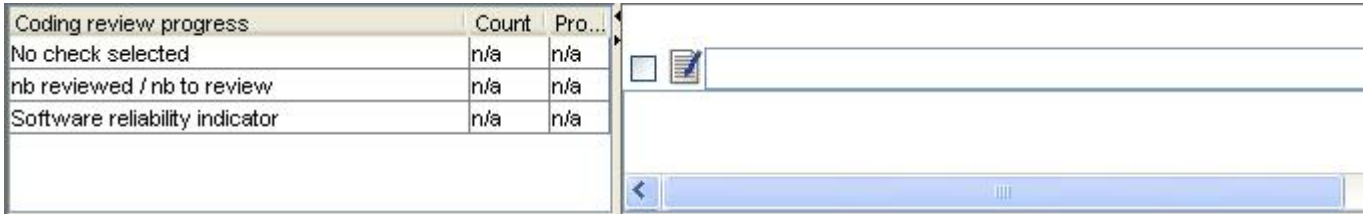
### Overview of Expert Mode

After loading the results, and PolySpace Viewer window looks like below:

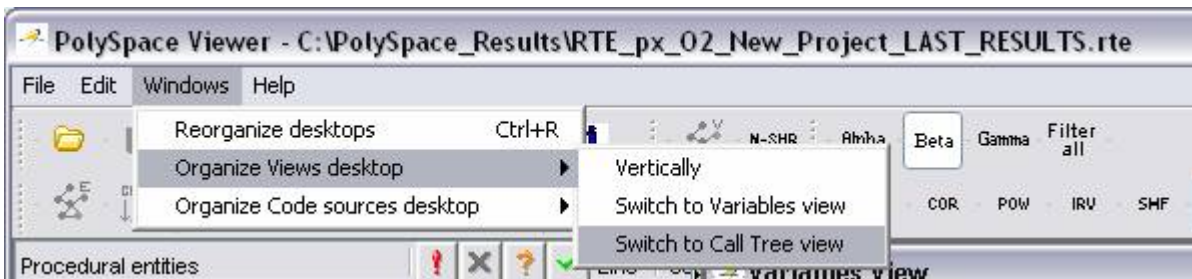


- On the left is the Procedural entities view (or RTE view). It displays the list of packages which have been analyzed or used during the analysis (specifications).
- In the bottom right area is the source code view with colored instructions. Each operation checked is displayed using meaningful color scheme and related diagnostic:
  - **Red** — Errors which occur at every execution.
  - **Orange** — Warning - an error may occurs sometimes.
  - **Grey** — Shows unreachable code.

- **Green** — Error condition that will never occur.
- The two windows just below the tool bar concern details of a currently reviewed check (when the check has been selected):



- The top right area is used for displaying both control and data flow results. You can switch from one view to the other by using the “Windows” menu:



## Procedural Entities View (RTE View)

Each file and underlying functions in the procedural entities view (or RTE view) is colored according to the most critical error found:

- `exception.stdh` — This file contains no check. This file contains stubs of the `<exception>` template part of the standard `stl` library. This template stubs is an accurate representation of the initial template provided by PolySpace. All templates of standard library have been stubbed to speed up analyses.
- `new.stdh` — This file contains no check. This file contains implemented stubs of `<new>` part of `stl` library template.

`__polyspace_main.cpp` — This file contains the main which was automatically generated. All checks there are green: no run-time error (or RTE) has been found. Please note that the pseudo code in this file is only


here to give information about the generated main. It must not be analyzed with PolySpace.

`training.cpp` — This file is red. This is the famous `training.cpp` containing the analyzed Class “MathUtils”. One or more *definite* run-time errors have been found in it.

`training.h` — This file is the famous `training.h`, locale header included in `training.cpp`. All checks are green: no run-time error (or RTE) has been found.

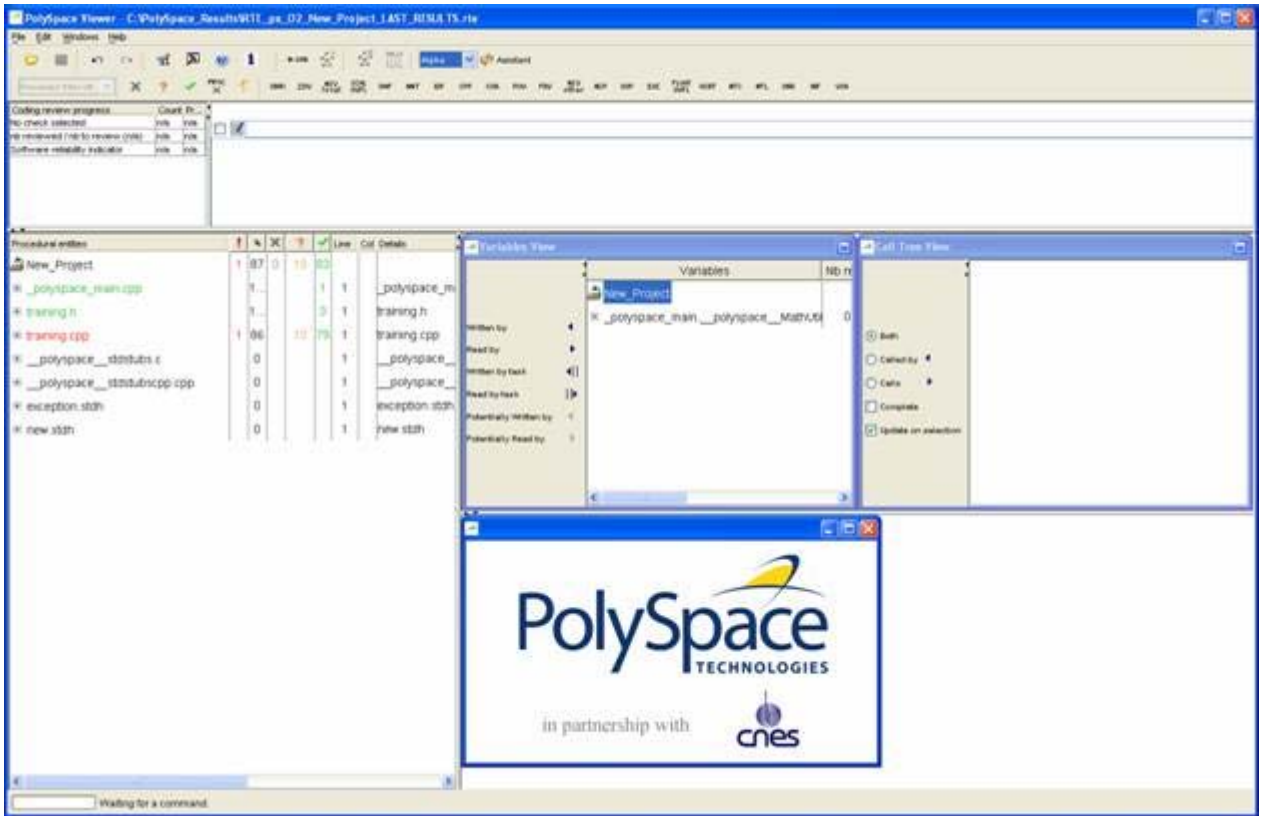
`__polyspace_stdstubs.c` — It contains stubs of standard functions part of `libc` library used in `training.cpp`. This file contains no check.






`__polyspace__stdstubs.cpp` — It contains stubs of some standard functions part of the `stl` library used in `training.cpp`. This file contains no check.



Click once on the  left of `training.cpp` to find out more about this file.




`training.cpp` is expanded and the list of function members defined within “MathUtils” of `training.cpp` is displayed. The function members in red or grey have code sections that need to be inspected (`MathUtils::Pointer_Arithmetic()`, `MathUtils::Recursion_caller()`, etc.) first because they are definite diagnosis of PolySpace (either runtime errors or dead code).






The columns (, , , , and ) provide information about run-time errors found in each function. The following table describes each of these columns.

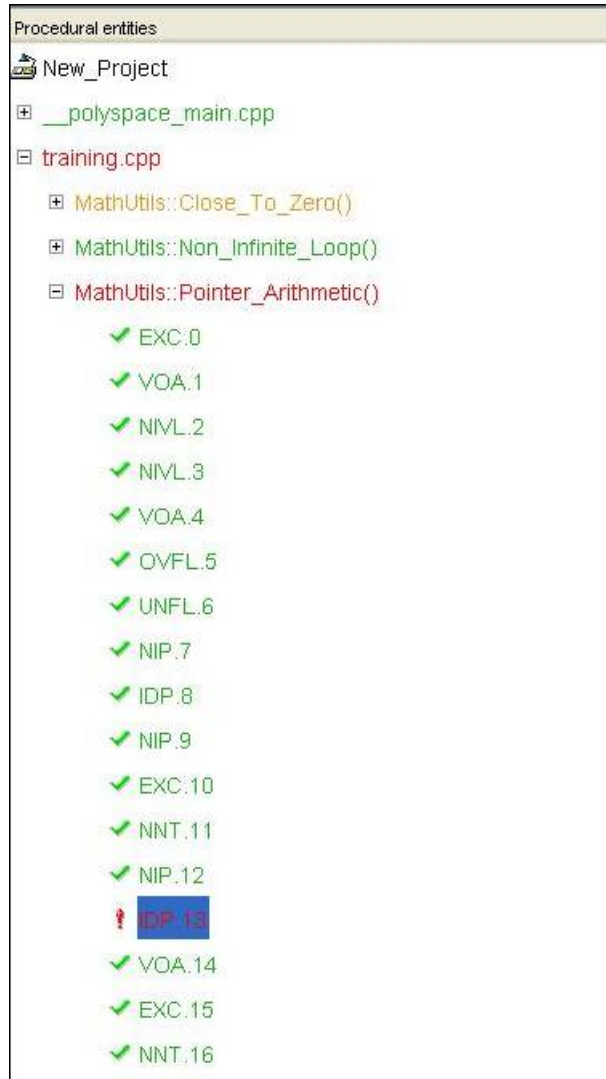
Column	Indicates
	Reliability of the code (level of proof).
	Number of definite run-time errors or reds.

Column	Indicates
	Number of warnings or oranges (that may hide run-time errors that do not occur systematically).
	Number of safe operations or greens.
	Number of unreachable instructions or grey code sections.

Let's have a look at some errors found by PolySpace in `training.cpp`.

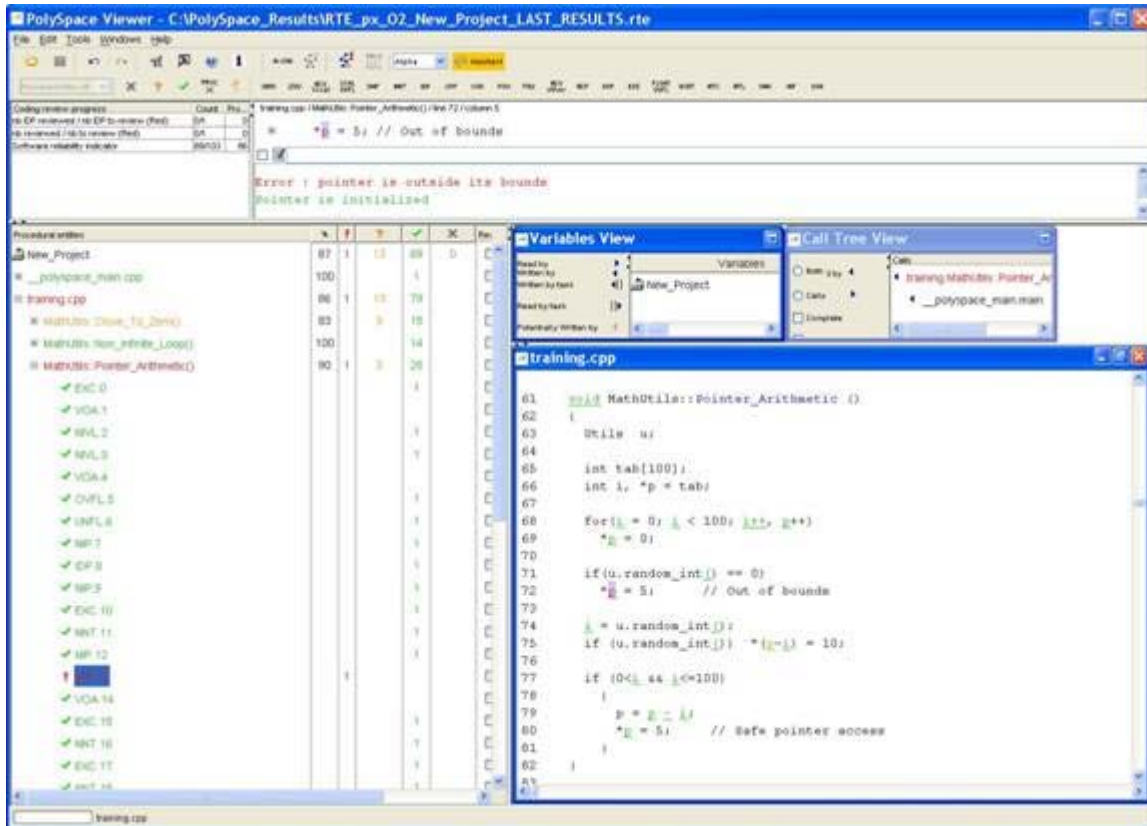
### First Example of Runtime Error Found by PolySpace: Memory Corruption

- 1 Click on  to expand “MathUtils::Pointer\_Arithmetic()“ to find out more about the red error. It displays a list of red, green, and orange symbols, featuring the complete list of code areas that PolySpace checked within the “MathUtils::Pointer\_Arithmetic()” function.

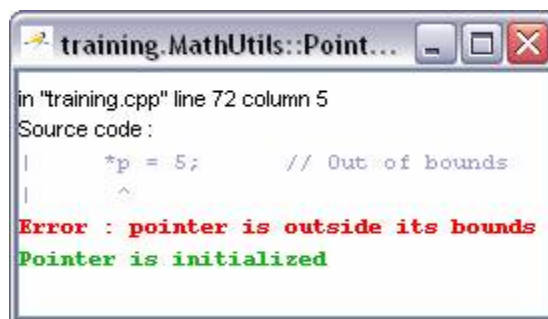


- 2 Click on the red “IDP.13” item - which stands for Illegal De-referenced Pointer -, to precisely locate this error in the source code. The bottom right section is updated showing the location of the “IDP.13” item.


## 4 PolySpace™ Viewer – Exploring Results

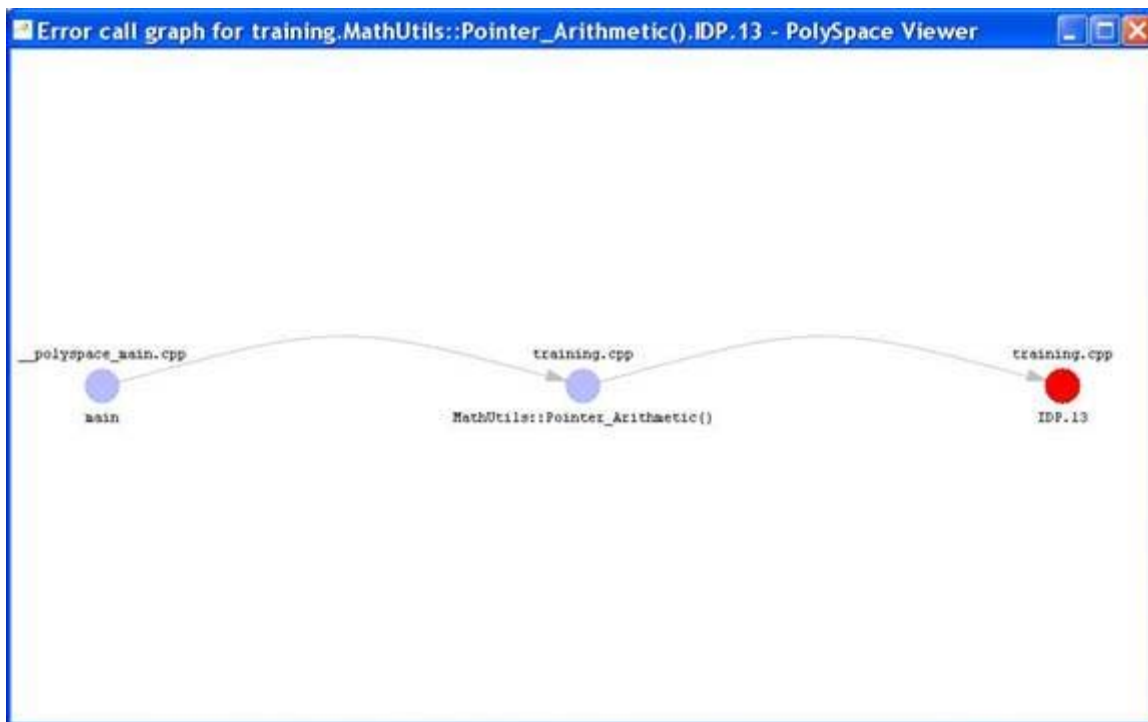


- 3 Click on red symbol in the source code at line 72. An error message is opened with the exact location:



Pointer `p` is de-referenced outside of its bounds. Indeed, at line 72 the instruction “`*p = 5;`” corrupts the memory as it puts the value “5” outside of the array “`tab`” pointed to by the pointer “`p`”.

- 4 You can also see the calling sequence leading to that particular red code section. To do so, select “IDP.13” item in the “Procedural entities” column in the RTE View, and then click on the  icon (on the top left of the PolySpace Viewer window) to display the corresponding run-time error access graph:



## Colors in the Source Code View

Each operation checked is also displayed using meaningful color scheme and related diagnostic in the source code view as links:

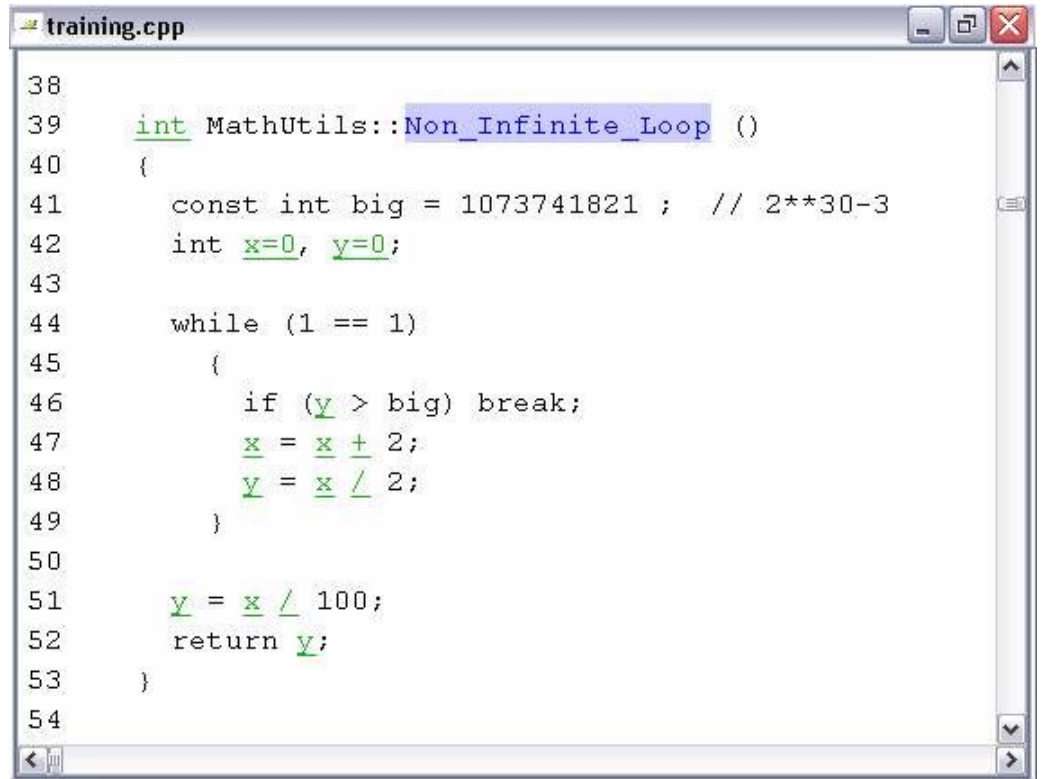
- **Red** — A link to the error message associated to the error which occurs at every execution.
- **Orange** — A link to an unproven message - an error may occur sometimes.
- **Grey** — A link to a check shown as unreachable code. The error message is in grey.
- **Green** — A link to a VOA (Value on Assignment) or an error condition that will never occur.
- **Black** — Represents some comments, source code that does not contain any operation to be checked by PolySpace in terms of run time errors and optimized operations, e.g. `x := 0;`
- **Blue** — Text highlighting the keyword “procedure” and “function”
- **Blue Underlined** — A link to a global variable in the “Global variable View”.

### More Examples of Run-Time Errors

Unlike most other testing techniques, PolySpace provides the benefit of finding the exact location of run-time errors in the source code. Below are some examples that you can review with PolySpace Viewer.

#### Example: Non-Infinite Loop

Select “MathUtils::Non\_Infinite\_Loop()” in the “Procedural entities” column in RTE View. The function is fully green: it means that the locale variable `x` never overflows, even if the exit condition of loop deals with `y` that is smaller than `x`. PolySpace confirms that the function always terminates.



```
38
39  int MathUtils::Non_Infinite_Loop ()
40  {
41      const int big = 1073741821 ; // 2**30-3
42      int x=0, y=0;
43
44      while (1 == 1)
45      {
46          if (y > big) break;
47          x = x + 2;
48          y = x / 2;
49      }
50
51      y = x / 100;
52      return y;
53  }
54
```

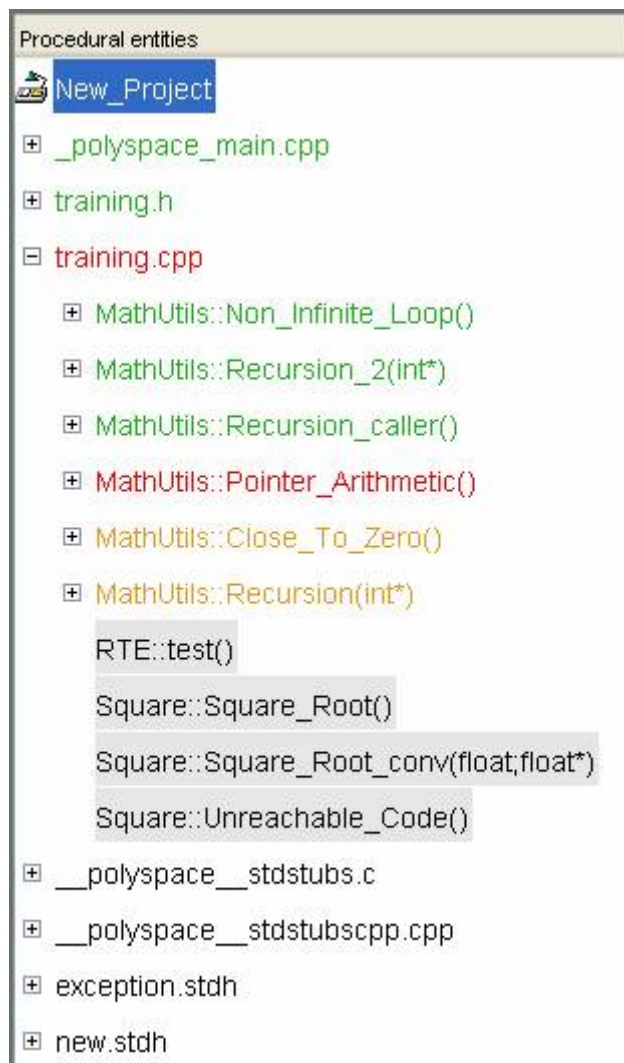
---

**Note** Using -voa option at launching time, PolySpace can help more suitably by giving information of range on scalar assignment.

---

### Example: Unreachable Code

We can also see in the “Procedural entities” column that some function members are never called. It is materialized by a reverse video in grey:






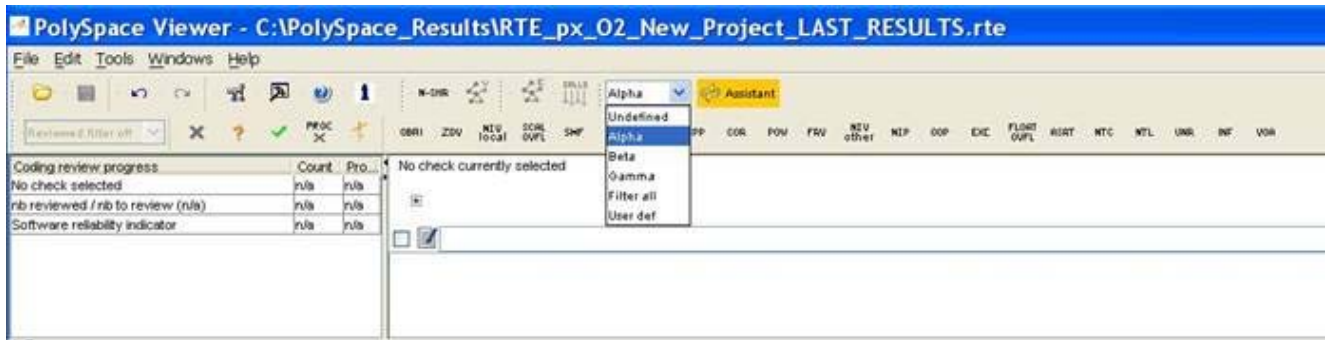
In the figure above it is the case for all public and protect member functions of “Square” and “RTE” classes. Indeed, the PolySpace analysis was made for the class “MathUtils”.



## Advanced Results Exploration

You can filter the information provided by PolySpace to focus on the type of errors you wish to investigate.


There are pre-defined composite filters (  ,  and  that you can choose depending on your development process. These filters are accessible through a combo list:



To illustrate the use of these filters, we will focus on the Pointer arithmetic member function that we have examined in a previous section.

### Gamma Mode

Gamma mode provides all the “red” and “grey” code sections. It is mainly used during the earliest development stages to focus quickly on critical bugs.

To select Gamma mode, click the  button.

The software reduces the information checks related to “MathUtils::Pointer\_Arithmetic()”.



This list of acronyms - for type of operations checked - shows what PolySpace automatically analyzed for you. In the case of member function is an illegal dereference pointer error (IDP.13).

---



**Note** VOA check (Value On Assignment) is only informative check that are never hidden.

---

### Beta Mode

Beta mode highlights checks that could cause a processor halt, memory corruptions or overflows. Beta mode is the default mode.


To select Beta mode:

- 1 Click .
- 2 Select “MathUtils::Pointer\_Arithmetic()” in the “Procedural entities” view.
- 3 Click  to get the list of the checks.

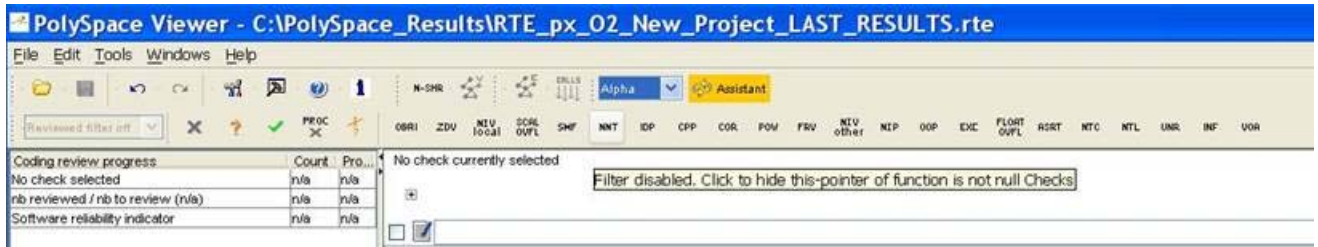


### Alpha Mode



Alpha Mode provides a comprehensive list of operations checked by PolySpace.

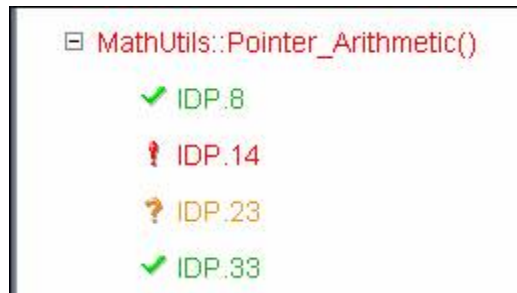
To switch to Alpha mode, click .


You may also want to use filters to focus on particular categories of errors. Those filters are located at the top of the PolySpace Viewer window:



**Note** When the mouse pointer moves on the filter, a tool tips gives its definition.

- Click  (top of the window) to suppress all checks, then click  .  
You will get list of checks containing only IDP (Illegal Dereference Pointers) reds, oranges or greens:








- Click  (top of the window) to suppress green code sections.  
You will get a reduced list of checks reds, oranges and grays:



## C++ specific checks

Specific C++ checks are dispatched in five categories:

- NNT category or Non Null This pointer (). It checks this pointer validity.
- CPP category. It concerns C++ related constructions () , like positive array size verification, *dynamic\_cast*, and typeid.
- OOP category. It concerns all C++ object oriented verification (): inheritance and virtual calls.
- EXC category. It concerns all C++ constructions dealing with exceptions ().
- INF category. It concerns information about C++ implicit and called functions when dealing with virtual functions ().

When reviewing C++ code with PolySpace Viewer, it is important to have a selective review check by check which follows the list of categories located at the top of the PolySpace Viewer window. Checks are classified from the left to the right. It is important to begin a review following this order. It is also important to begin by C “like” checks before C++ like “checks”.

This methodology permits to focus first by the categories which are most susceptible to hide run time errors. This methodology has been automatically applied in the “Methodological Assistant” on page 4-25.

## Miscellaneous

The  icon gives access to the PolySpace Manual. All views have a pop-up menu (right click on mouse).

Close the PolySpace Viewer window by clicking on the upper right



symbol (PolySpace Viewer can also be closed using **File > Close**).

# Methodological Assistant

## In this section...

“Methodological Assistant Overview” on page 4-25

“Opening the Methodological Assistant ” on page 4-25

“Assistant Dashboard” on page 4-26

“Choose a Methodological Assistant” on page 4-30

## Methodological Assistant Overview

After a first navigation into the PolySpace Viewer, some simple questions remain:

- Do all checks need to be reviewed?
- What are the checks to review?
- How many?
- What is the best order?

The Methodological assistant is here to answer to all these questions: It helps to select and manage the checks to be reviewed. It selects a “best” subset and sorts out them. The Assistant mode in the PolySpace Viewer will then guide through these selected checks.

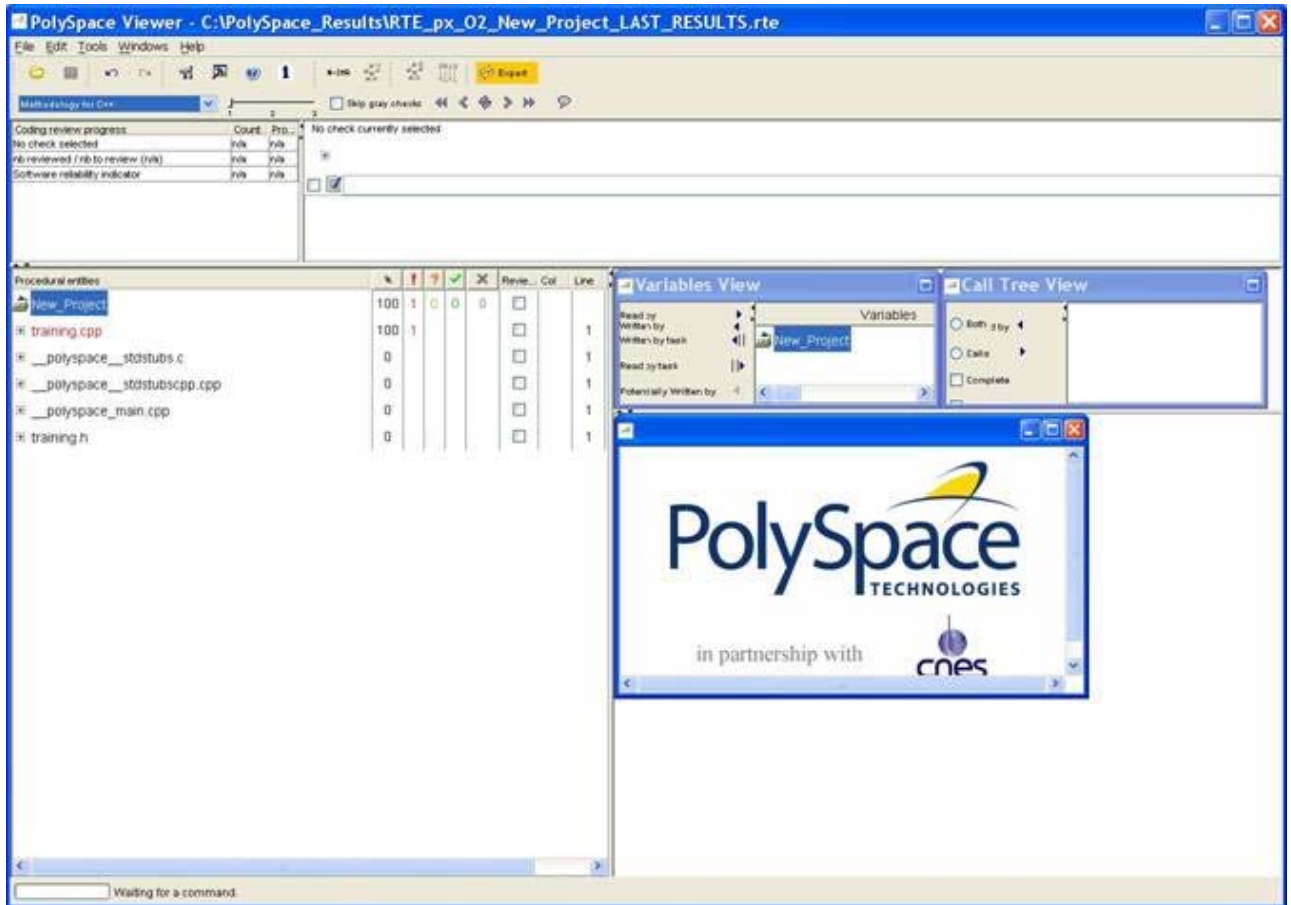
## Opening the Methodological Assistant

To open the assistant:

- 1 If the PolySpace Viewer is still open, close it.
- 2 Open the PolySpace Viewer again, then load the same results.
- 3 Choose “Assistant” mode.

After having loaded the results in “Assistant” mode, PolySpace Viewer window looks like below:

## 4 PolySpace™ Viewer – Exploring Results



### Assistant Dashboard


The second line of buttons on the toolbar and the two views just below are the navigation centre based on the methodological method used in the assistant mode:





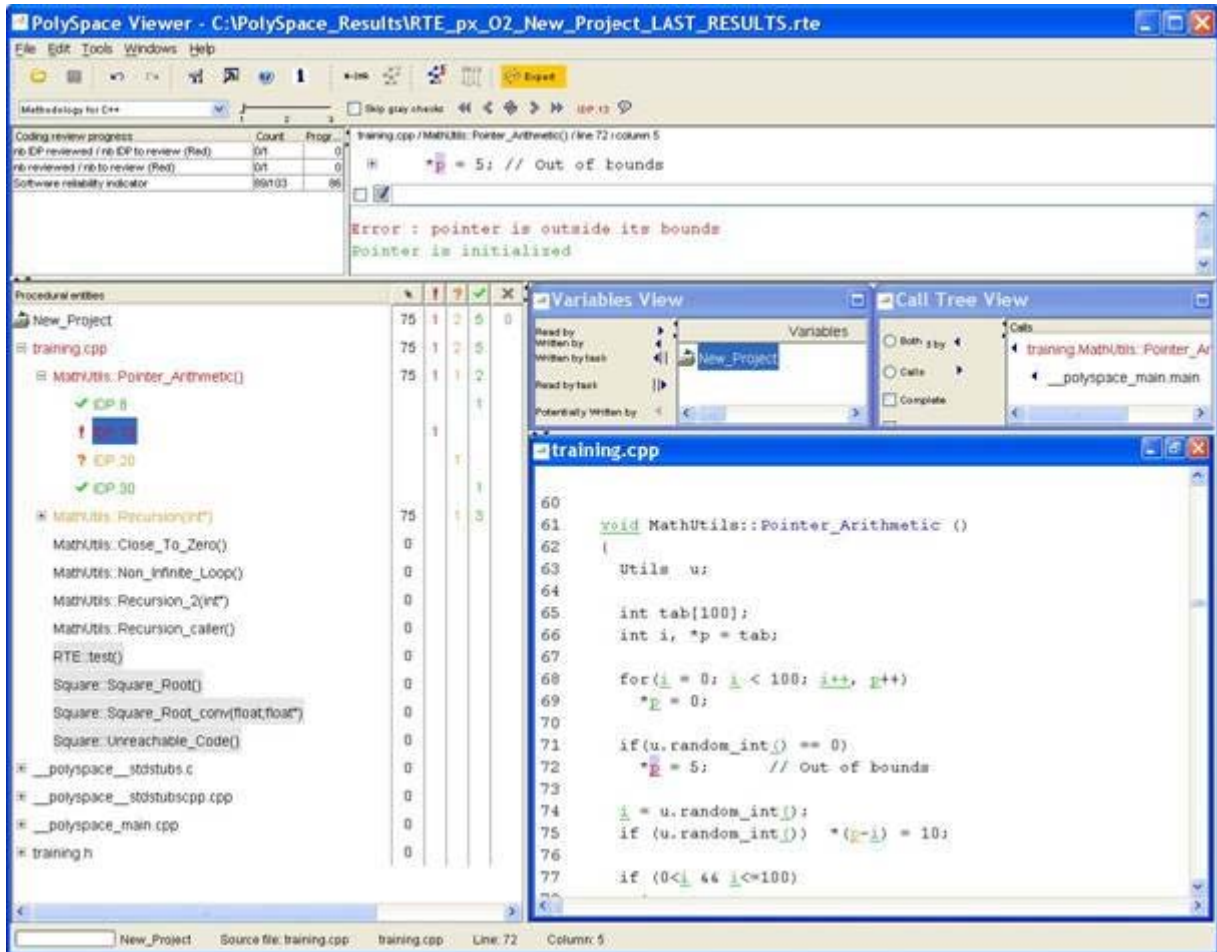
Some other changes can be seen in the viewer:

- Now, in the “Procedural Entities” view the list of files analyzed *is sorted by the methodological assistant used*.
- In the bottom right area is the source code view with colored instructions. Each operation will be checked and sorted by the methodological method using meaningful color scheme and related diagnostic and in the following order:
  - **Red** — Assistant browses all errors which occur at every execution.
  - **Gray** — Assistant browses each block of unreachable code depending if radio button “Skip gray checks” has been ticked or not.
  - **Orange** — Assistant chooses and reviews the “best” unproven operations -errors that may occur sometimes.

- 1 Click  to navigate to next check.

The PolySpace Viewer is refreshed with the first check selected by the Methodology of review:

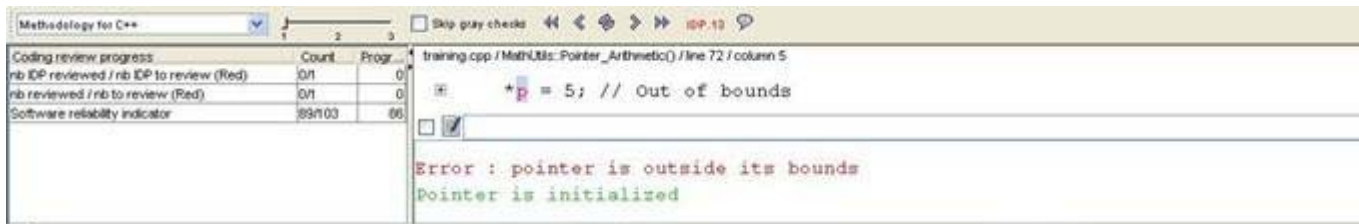
## 4 PolySpace™ Viewer – Exploring Results



The Methodological dashboard gives details and allows reviewing the check. On the selected check, it is possible to mark the fact that it has been reviewed.

- 2 Select the radio button box.
- 3 Enter a comment in the associated edit box on the right.

After, it looks like:



The left part of the dashboard has been updated, and displays some statistics in three lines:

- The first line gives the number and percentage of remaining checks to review of the current category. In the previous example, it concerns red IDP checks.
- The second line gives values in the color category (red, grey and unproven).
- The Last line gives in permanence the Software reliability indicator.

Other buttons in the Methodological dash board allow navigating to previous check, coming back to current one



and going to next



/ previous



category of reviewed checks selected by the Methodology.

## Choose a Methodological Assistant



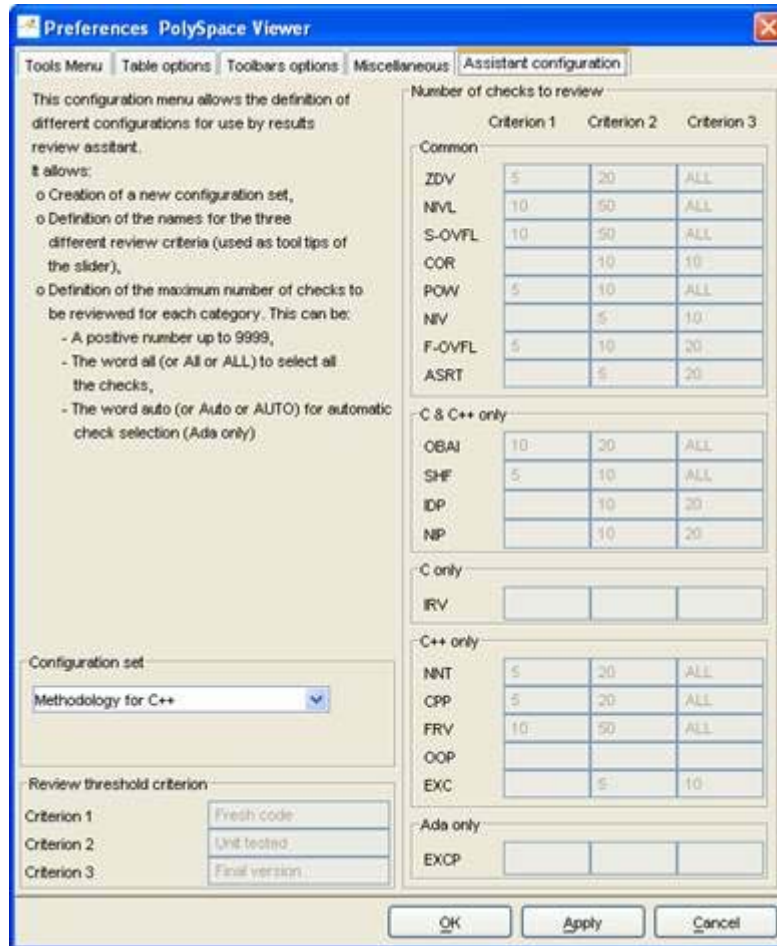
have been pre-selected by PolySpace.

The methodology allows selecting the categories of checks to review, the number for each category and their order depending of a statistical algorithm.

The level (or criterion) defines the number of checks to review by category. Explicit names have been associated to each criterion like “Fresh code”, “Unit test” and “Code review”

It is possible to refine a self-created one or define its own Methodology.

- 1 Select **Edit > Preferences** in the PolySpace Viewer.
- 2 Select the Assistant Configuration tab.



### 3 Create a new configuration set.

Define the categories of check to review for each criterion, how many in each one.

---

**Note** You cannot change an existing configuration except by duplication and refinement.

---

## Report Generation

When PolySpace performs an analysis, it generates textual files that can be used to generate Microsoft® Excel® reports.

---

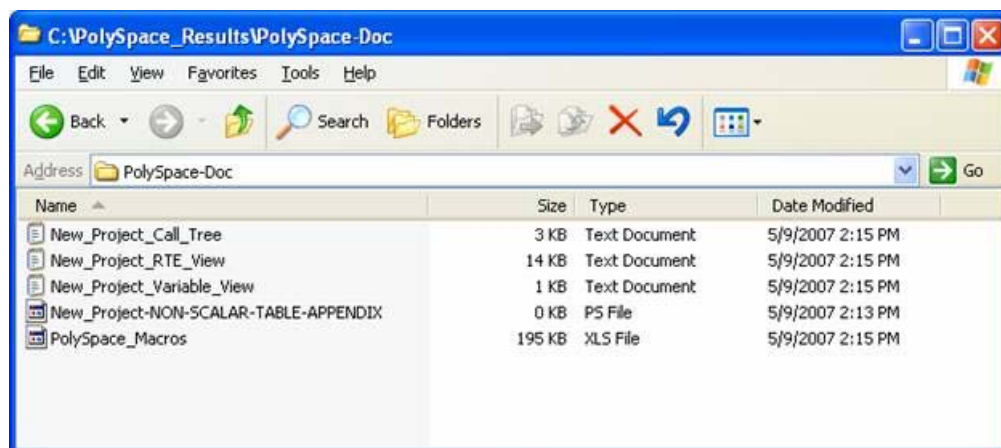
**Note** Excel report is an option of PolySpace Desktop and Verifier only available under license. If you do not currently have a license and would like to learn more about it, please contact The Mathworks.

---

These files are located in the results directory (see C:\PolySpace\_Results\PolySpace-Doc or *PolySpaceInstallDir*\Examples\Demo\_CPP\PolySpace-Doc).

All views (except source code) are printable and can be exported to textual or Excel format (protected by license).

The C:\PolySpace\_Results\PolySpace-Doc directory should contain the following files:



To generate a report:

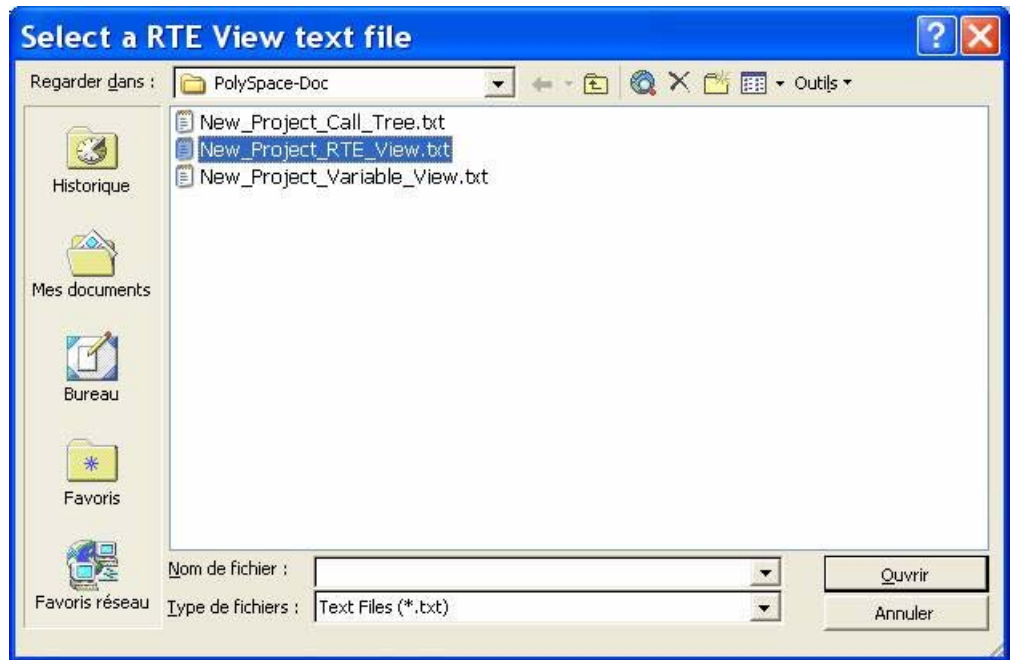
- 1 Open the file called `PolySpace_Macros.xls`, enable macros when asked and then the following window opens:

	A	B	C	D	E	F	G	H
1								
2	Copyright © PolySpace Technologies, 1999-2006							
3								
4	<div style="border: 1px solid black; padding: 10px;"> <div style="display: flex; justify-content: space-between;"> <div style="border: 1px solid black; padding: 5px;"> <p>Apply filters?</p> <p><input checked="" type="radio"/> No filters</p> <p><input type="radio"/> Beta filters</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p>Generate checks by file?</p> <p><input checked="" type="radio"/> yes</p> <p><input type="radio"/> no</p> </div> </div> <p style="text-align: center;"> <span style="border: 1px solid black; padding: 2px 5px;">Help</span> <span style="margin: 0 20px;">Use this button to create the complete synthesis in one file. Select the RTE export view and a file in which to save results. If the other views are in the same directory as the RTE view then they will automatically be incorporated into the same file.</span> <span style="border: 1px solid black; padding: 2px 5px;">Help</span> </p> <p style="text-align: center; color: red; font-weight: bold;"> <span style="border: 1px solid black; padding: 5px 15px;">Generate PolySpace Results Synthesis</span> </p> </div>							
9								
10								
11								
12								
13								
14								
15								
16								
17	Reports can be generated from all PolySpace txt file format results. These are generated							
18	by the PolySpace Verifier during an analysis, the export option in the PolySpace Viewer,							
19	or from the command line using the "gen-excel-files" command.							
20								
21	Individual PolySpace text result files can be processed using the below macros:							
22								
23	<u>The macros are:</u>							
24	<div style="display: flex; justify-content: space-between;"> <div style="border: 1px solid black; padding: 2px 10px; background-color: #e0e0e0;">RTE</div> <div>Apply to RTE views exported from PolySpace Viewer</div> </div>							
25	<div style="display: flex; justify-content: space-between;"> <div style="border: 1px solid black; padding: 2px 10px; background-color: #e0e0e0;">Call Tree</div> <div>Apply to Call Tree views exported from PolySpace Viewer</div> </div>							
26	<div style="display: flex; justify-content: space-between;"> <div style="border: 1px solid black; padding: 2px 10px; background-color: #e0e0e0;">Variables</div> <div>Apply to Variable views exported from PolySpace Viewer</div> </div>							
27								
28								
29	Version 3.4.1D				RTE = Run Time Error			
30								

2 Click on

Generate PolySpace Results Synthesis

A file browser opens.



**3** Select the file called `New_Project_RTE_View.txt`.

After a few seconds, an Excel file is generated. It contains several spreadsheets related to the application analyzed.

Application Call Tree / Shared Globals / Global Data Dictionary / Checks by file / Check Synthesis / Launching Options / RTE -> All checks location / Orange C

For example, in “Checks Synthesis” all statistics about checks and colors are reported in a summary table.



	A	B	C	D	E	F	G
1	<b>RTE Statistics</b>						
2	<b>Check category</b>	<b>Check detail</b>	<b>R</b>	<b>O</b>	<b>Gy</b>	<b>Gr</b>	<b>% proved</b>
3	OBAI	Out of Bounds Array Index	0	0	0	0	0,00%
4	NIVL	Uninitialized Local Variable	0	0	1	28	100,00%
5	IDP	Illegal Dereference of Pointer	1	1	0	7	88,89%
6	NIP	Uninitialized Pointer	0	0	0	12	100,00%
7	NIV	Uninitialized Variable	0	0	0	8	100,00%
8	IRV	Initialized Value Returned	0	0	0	15	100,00%
9	COR	Other Correctness Conditions	0	0	0	2	100,00%
10	ASRT	User Assertion Failure	0	0	0	0	0,00%
11	POW	Power Must Be Positive	0	0	0	0	0,00%
12	ZDV	Division by Zero	0	1	0	4	80,00%
13	SHF	Shift Amount Within Bounds	0	0	0	0	0,00%
14	OVFL	Overflow	0	3	2	8	76,92%
15	UNFL	Underflow	0	1	2	9	91,67%
16	UOVFL	Underflow or Overflow	0	3	0	4	57,14%
17	EXCP	Arithmetic Exceptions	0	0	0	0	0,00%
18	NTC	Non Termination of Call	3	0	0	0	100,00%
19	k-NTC	Known Non Termination of Call	0	0	0	0	0,00%
20	NTL	Non Termination of Loop	0	0	0	0	0,00%
21	UNR	Unreachable Code	0	0	0	0	0,00%
22	UNP	Uncalled Procedure	0	0	0	0	0,00%
23	IPT	Inspection Point	0	0	0	0	0,00%
24	OTH	other checks	0	0	0	0	0,00%
25	Total :		4	9	5	97	92,17%

This ends ways of results review.



# Launch PolySpace™ Remotely

---

Overview (p. 5-2)	Provides an overview of the exercise performed in this chapter
Launching an Analysis (p. 5-3)	Describes how to perform an analysis using PolySpace™ Server™ for C/C++
Managing Your Remote Analysis: the PolySpace™ Spooler (p. 5-5)	Describes how to manage remote analyses
Batch Commands (p. 5-8)	Describes how to perform and manage analyses in Batch
Sharing Analyses Between Accounts (p. 5-11)	Describes how to share the results of a Server analysis between multiple Client users

### Overview

This chapter describes the basic steps to launch an analysis in remote.

To do so you need:

- A Queue Manager server (QM) installed.
- Your desktop PC configured with the PolySpace™ Client™ for C/C++ product.
- A networked machine configured with the PolySpace™ Server™ for C/C++ product.

Please see the PolySpace Installation Guide (available on the PolySpace CD-ROM in \Docs\Install or the PolySpace Install Guide Manual) to install and configure a Client and a Server.


---

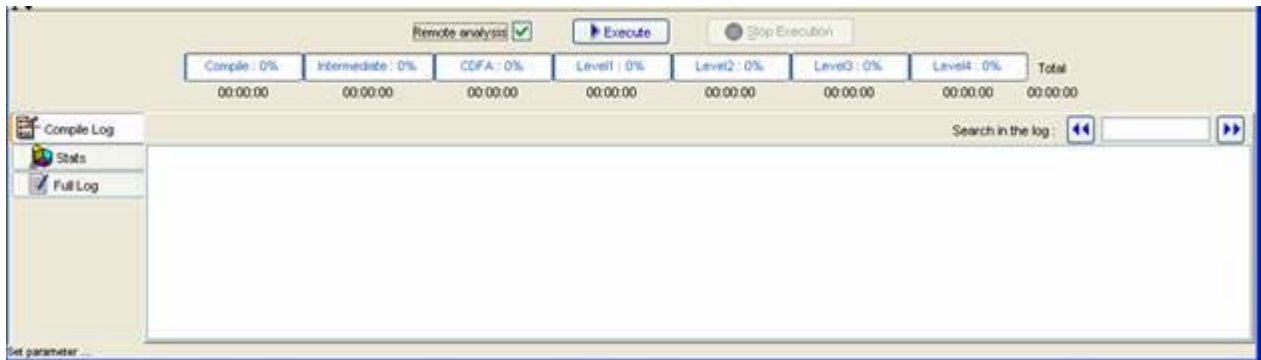
**Note** Launching an analysis remotely requires a PolySpace Server for C/C++ product and associated license.

---

## Launching an Analysis

To launch PolySpace remotely:

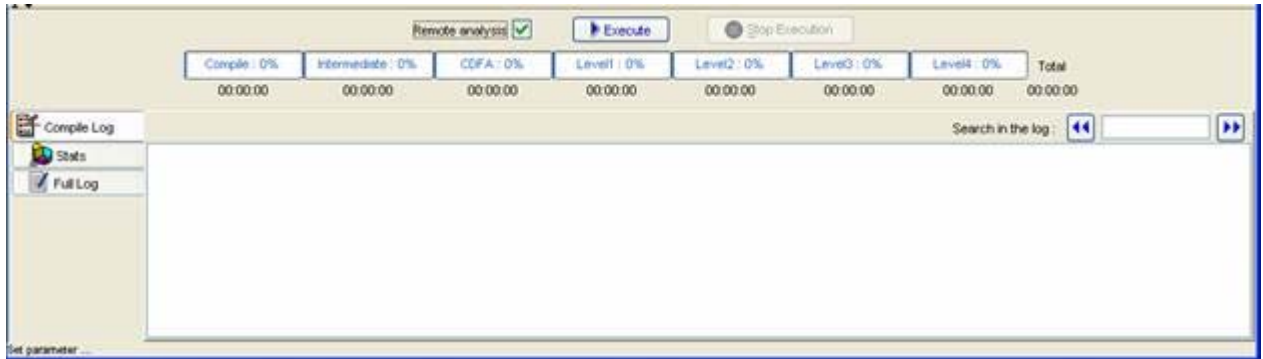
- 1 Set up an analysis as described in Chapter 2, “PolySpace™ Client – Setting Up and Launching an Analysis on a Single Class”, but do not launch it.
- 2 Select the **Remote analysis** checkbox (see next figure).
- 3 Click  to launch the analysis.



The analysis starts and the compilation phase is performed on the desktop PC. At the end of the “C source verification phase” the analysis is sent to the Queue Manager server.

- 4 Click on the **Full Log** tab. You will have a message like this:

## 5 Launch PolySpace™ Remotely



The analysis has been queued with an ID number, and you can follow its progression using the PolySpace Spooler.

---

**Note** If you do not select the “Remote analysis” radio button, the analysis continues locally.

---

## Managing Your Remote Analysis: the PolySpace™ Spooler

You can check the analysis processes in the queue using the PolySpace™ Spooler.

To manage an analysis in the queue:

- 1 Open the PolySpace Spooler by either:
  - Clicking on the short cut on your desktop PC



- Clicking on the icon

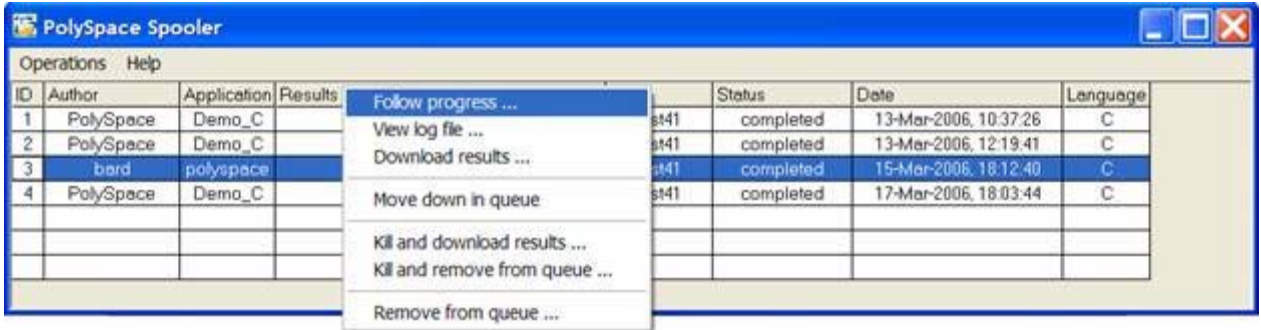


in the menu tab of the launcher.

The PolySpace Spooler appears.

ID	Author	Application	Results directory	CPU	Status	Date	Language
1	PolySpace	Demo_C	c:\Demo_C_results	test41	completed	13-Mar-2006, 10:37:26	C
2	PolySpace	Demo_C	c:\RESULTS\results_voa	test41	completed	13-Mar-2006, 12:19:41	C
3	bird	polyspace	c:\PolySpace\B3\Venter	test41	completed	15-Mar-2006, 18:12:40	C
4	PolySpace	Demo_C	c:\RESULTS\results_voa	test41	completed	17-Mar-2006, 18:03:44	C

- 2 Right-click on an analysis to manage it in the queue:



### 3 Select one of the following options:

- **Follow progress** — This action lists the associated log file in a Launcher window. If the analysis is running, you can follow the update of the log file and associated progress bar in real time on the Launcher window.
- **View log file** — This action lists the associated log file in a “Command prompt” window, in which you can the last 100 updated lines of the log file in real time. This option is only available when the analysis is running.
- **Download results** — This action downloads the results of an analysis onto the client. If the analysis is still running, available results are downloaded on the client, without disturbing the analysis. The option is not possible for a “queued” analysis.
- **Move down in queue** — This action reduces the priority of a “queued” analysis.
- **Kill and download results** — This action stops the analysis definitively and the results are downloaded. The status of the analysis changes from “running” to “aborted”. The analysis remains on the queue.
- **Kill and remove from queue** — This action stops the analysis definitively, and the analysis is removed from the queue.

---

**Caution** The results will be lost.

---



- **Remove from queue** — This action removes a “queued”, “aborted” or a “completed” analysis.

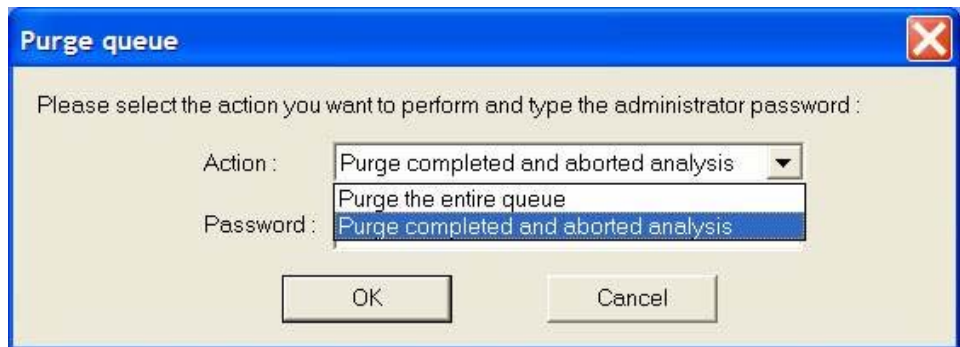
---

**Caution** The results will be lost.

---

You can also manage the queue from an administrator point of view using the **Operations** menu:

- Select **Operations > Purge queue**, to purge the entire queue or purge only completed and aborted analysis (see next figure).



---

**Note** The queue manager password is required.

---

- Select **Operations > Change root password**, to change the administrator password.

---

**Note** By default this password does not exist.

---

**On a UNIX® platform**, there is no graphical user interface but a set of “Batch Commands” on page 5-8 which allow the management of analyses on the queue. All these commands begin with the prefix `PolySpaceCommonDir/RemoteLauncher/psqueue -.`

# Batch Commands

In this section...
“Launching an Analysis in Batch” on page 5-8
“Managing an Analysis in Batch” on page 5-8

## Launching an Analysis in Batch

A set of commands allow you to launch an analysis in batch.

All these commands begin with the following prefixes:

- **Server analysis** —  
*PolySpaceInstallDir/Verifier/bin/polyspace-remote-cpp*
- **Client analysis** — *polyspace-remote-desktop-cpp*

These commands are equivalent to commands with a prefix *PolySpaceInstallDir/bin/polyspace-*.

For example, *polyspace-remote-desktop-cpp -server [<hostname>:[<port>] | auto]* allows you to send a C++ client analysis remotely.

---

**Note** If your PolySpace server is running on Microsoft® Windows®, the batch commands are located in the */wbin/* directory. For example, *PolySpaceInstallDir/Verifier/wbin/polyspace-remote-cpp.exe*

---

## Managing an Analysis in Batch

In batch, a set of commands allow the management of analysis in the queue.

On UNIX®, all these commands begin with the prefix *PolySpaceCommonDir/RemoteLauncher/bin/psqueue-*:

On Windows, these commands begin with the prefix *PolySpaceCommonDir/RemoteLauncher/wbin/psqueue-*:

- `psqueue-download <id> <results dir>` — download an identified analysis into a results directory.
  - `[-f]` force download (without interactivity)
  - `-admin -p <password>` allows administrator to download results.
  - `[-server <name>[:port]]` selects a specific Queue Manager.
  - `[-v|version]` gives release number.
- `psqueue-kill <id>` — kill an identified analysis.
- `psqueue-purge all|ended` — remove all or finished analyses in the queue.
- `psqueue-dump` — gives the list of all analyses in the queue associated to default Queue Manager.
- `psqueue-move-down <id>` — move down an identified analysis in the Queue.
- `psqueue-remove <id>` — remove an identified analysis in the queue.
- `psqueue-get-qm-server` — give the name of the default Queue Manager.
- `psqueue-progress <id>`: give progression of the currently identified and running analysis.
  - `[-open-launcher]` display the log in the graphical user interface of launcher.
  - `[-full]` give full log file.
  - `psqueue-set-password <password> <new password>` — change administrator password.
- `psqueue-check-config` — check the configuration of Queue Manager.
  - `[-check-licenses]` check for licenses only.
- `psqueue-upgrade` — Allow to upgrade a client side (cf. PolySpace Installation Guide in the `PolySpaceCommonDir/Docs` directory).
  - `[-list-versions]` give the list of available release to upgrade.
  - `[-install-version <version number> [-install-dir <directory>]] [-silent]` allow to install an upgrade in a given directory and in silent.

---

**Note** *PolySpaceCommonDir/bin/psqueue-<command>* -h gives information about all available options for each command.

---

## Sharing Analyses Between Accounts

### In this section...

“Analysis-key.text File” on page 5-11

“Magic Key or Shared Analysis Between Projects” on page 5-12

### Analysis-key.text File

From a security point of view, all analysis spooled on a same Queue Manager are owned by the user who sent the analysis from a specific account. Each analysis has a unique cryptic key.

The public part of the key is stored in a file `analysis-keys.txt` associated to a user account. This file is located in:

- **UNIX**<sup>®</sup> — `/home/username/.PolySpace`
- **Windows**<sup>®</sup> — `C:\Documents and Settings\username\Application Data\PolySpace`

The format of the ASCII file is the following (spaces are tabulation):

```
<id of launching> <server name of IP address> <public key>
```

where *<public key>* is a value in the range [0..F].

#### Example:

```
1 m120 27CB36A9D656F0C3F84F959304ACF81BF229827C58BE1A15C8123786
2 m120 2860F820320CDD8317C51E4455E3D1A48DCE576F5C66BEEF391A9962
8 m120 2D51FF34D7B319121D221272585C7E79501FBCC8973CF287F6C12FCA
```

When we make an attempt of management (download, kill and remove, etc.) on a particular analysis, the Queue Manager will examine this file and find the associated public key to authenticate the analysis on the server.

If the key does not exist, an error message appears: “key for analysis <ID> not found”. So sharing an analysis with another user account necessitates the public key.

Sharing an analysis is quite simple, ask to the owner of the analysis the line in `analysis-key.txt` which containing the associated `<ID>` and put it the line in your own file. After, it will be able to download the analysis.

### **Magic Key or Shared Analysis Between Projects**

A magic key allows sharing analyses without taking into account the `<ID>`. It allows same key for all analysis launched by a user account. The format is the following:

```
0 <Server id> <your hexadecimal value>
```

All analyses spooled will have this key instead of random one. In the same way, if this kind of key is available in an `analysis-key.txt` file of another user, it allows to authorize any operation on any analyses pushed with this key.

---

**Note** It only works for all analysis launched after having put the magic key in the file. If the analysis has been launched before, the allowed key associated to the ID will be used for the authentication.

---

# Summary

---

After having followed each step of this tutorial, you are now able to launch a class analysis using the PolySpace™ Client™ for C/C++ product, and explore some results with PolySpace™ Viewer. All these commands can be performed locally on your desktop PC or in client/server architecture.

You will find more information on advanced options available with our tools in the *PolySpace Client/Server for C++ User's Guide* available on the CD-ROM or in `PolySpaceCommonDir\Docs\Manuals`.