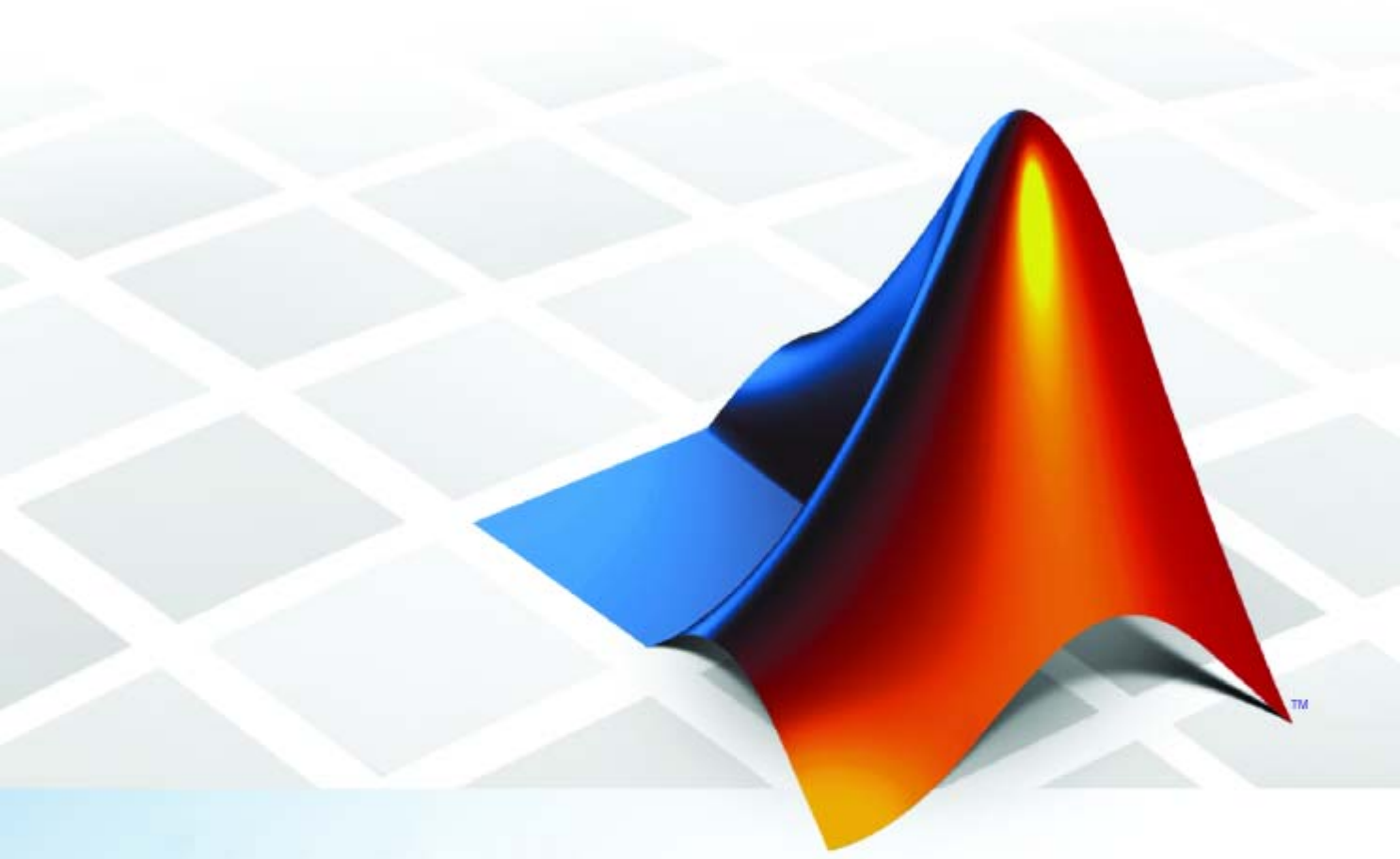
PolySpace™ Client/Server for C 5

Getting Started Guide



The MathWorks™

Accelerating the pace of engineering and science

How to Contact The MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

PolySpace™ Client/Server for C Getting Started Guide

Trademarks

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

Patents

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

Revision History

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

Contents

1 General Requirements

2 PolySpace™ Client — Analyzing a Single C File

PolySpace™ Viewer — Exploring Results

3

Setting Up and Launching the MISRA-C Checker

4

5 Launching PolySpace™ Analysis Remotely

6 Summary

1

General Requirements

Computer Configuration

Please refer to PolySpace™ Installation Guide for the minimum hardware requirements.

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.
- The fourth 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 — Analyzing a Single C File”.

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

- *PolySpace™ Client™ for C/C++* for analyzing single files. Note that this module is available with the icon “*PolySpace Launcher*”.
- *PolySpace™ Server™ for C/C++* for multi-file or composite analysis. Note that this module is available with the icon “*PolySpace Launcher*”.
- *PolySpace Viewer* is the graphical user interface to explore the results computed by PolySpace Server for C/C++ or PolySpace Client for C/C++.
- *PolySpace Spooler* is the graphical interface to manage analysis done remotely.

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:

Moreover, inside PolySpace™ Client™ for C/C++ and PolySpace™ Server™ for C/C++, a PolySpace MISRA® Checker is available, allowing at compilation time to verify some of the rules recommended by the MISRA Consortium (more about MISRA Consortium at `http://www.misra-c.com`).

This Getting Started Guide will focus on the following four exercises using the Client, the Viewer, the PolySpace MISRA Checker and the Server:

- In Chapter 2, “PolySpace™ Client — Analyzing a Single C File”, you will analyze a simple file “example.c” by using the PolySpace Client for C/C++ product.
- In Chapter 3, “PolySpace™ Viewer — Exploring Results”, you will review the results obtained in Chapter 2 by using PolySpace Viewer.
- In Chapter 4, “Setting Up and Launching the MISRA-C Checker”, you will use PolySpace MISRA Checker during the compilation phase of a PolySpace analysis.
- In Chapter 5, “Launching PolySpace™ Analysis Remotely”, you will send an analysis remotely to a PolySpace Server for C/C++ server.

2

PolySpace™ Client — Analyzing a Single C File

Overview

This chapter describes a basic file analysis. It focuses on the analysis of `example.c`, which is included in the PolySpace™ installation directory and located at:

`PolySpaceInstallDir``\Examples\Demo_C\sources\example.c`.

The PolySpace analysis process is composed of three main phases:

1. First, PolySpace software checks the syntax and semantic of the analyzed file(s). However, as PolySpace products are not associated with a particular compiler, **benefits** of this phase are triple for the analyzed source code: **ANSI® compliance**, **portability** and **maintainability**.
2. Then, the client seeks the main procedure. If none is found, PolySpace™ Client™ for C/C++ will generate one automatically. By default, this function will call all public functions of the file.
3. Finally, the client 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™ products and their related license files are correctly installed.
- Source code files (in this case `example.c`) and all header files that it may directly or indirectly include. For this tutorial we will see later that we need two header files `math.h` and `include.h` in order to analyze `example.c`.
- All “-D” compilation switches necessary to compile the file are known. Please note that in this tutorial, no “-D” is necessary to compile `example.c`.

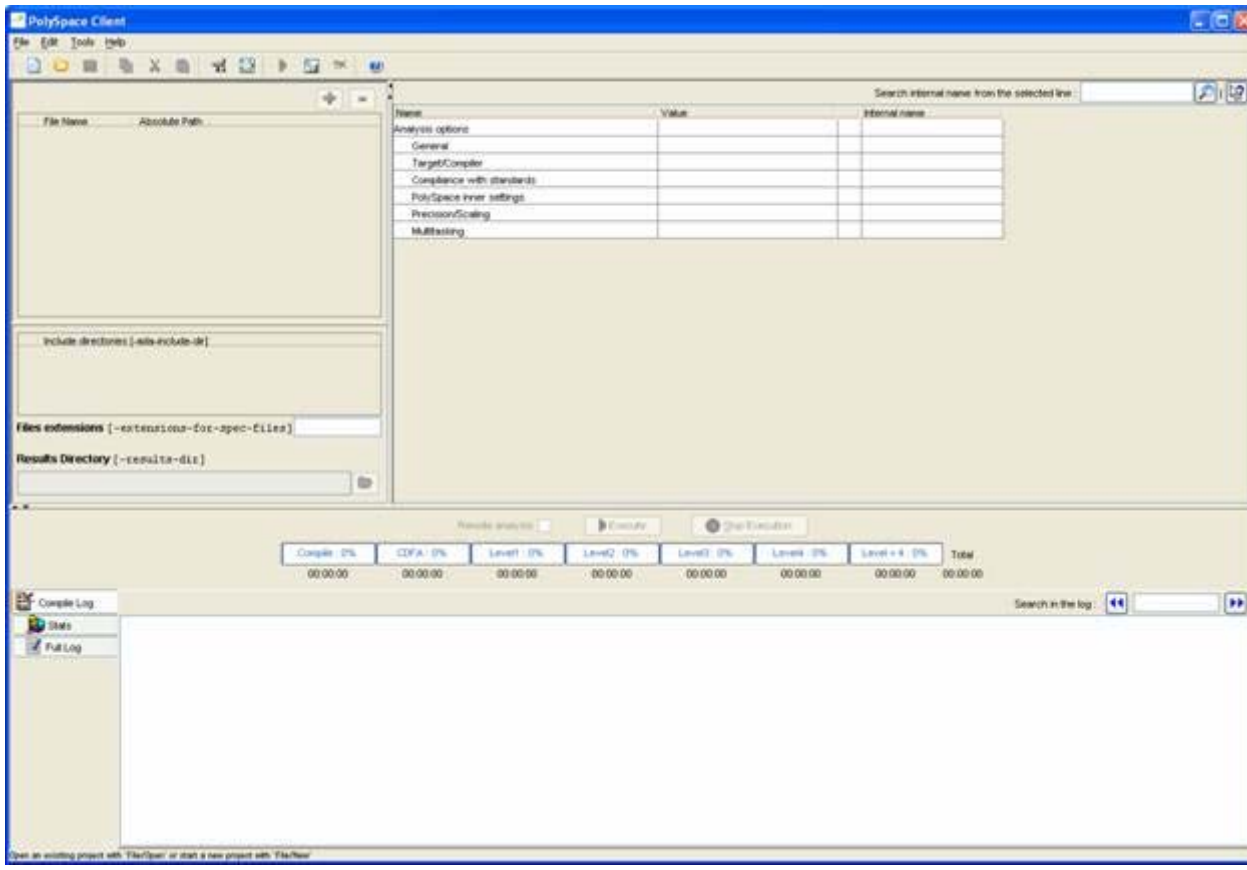
Setting Up a PolySpace™ Client™ for C/C++ Analysis

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 **Client Launcher**, and **C**, then click **OK**.

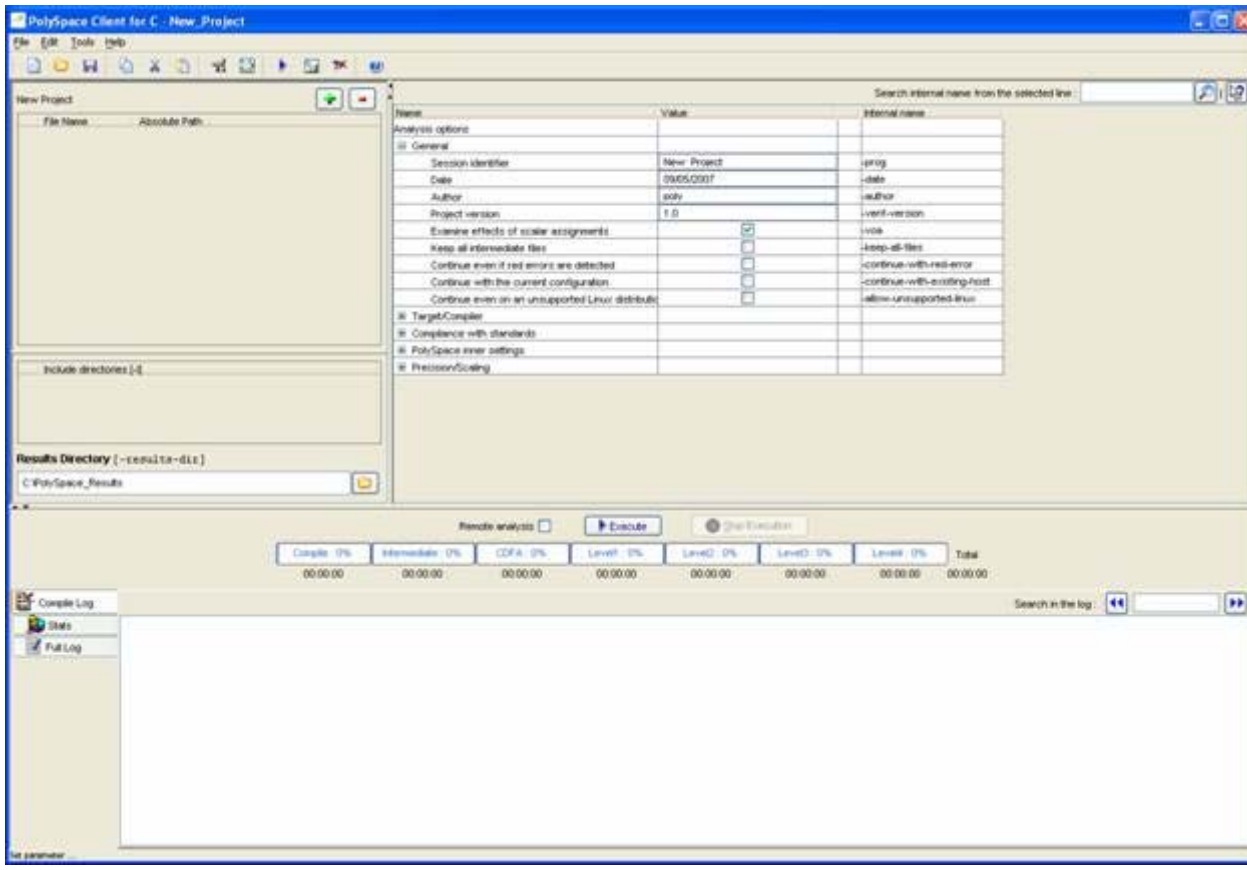
The Graphical Interface of PolySpace™ analysis Launcher is displayed as below:



3 Click on File/New Project to start an analysis:

The PolySpace™ Client™ for C/C++ New Project window opens (see figure below). 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.

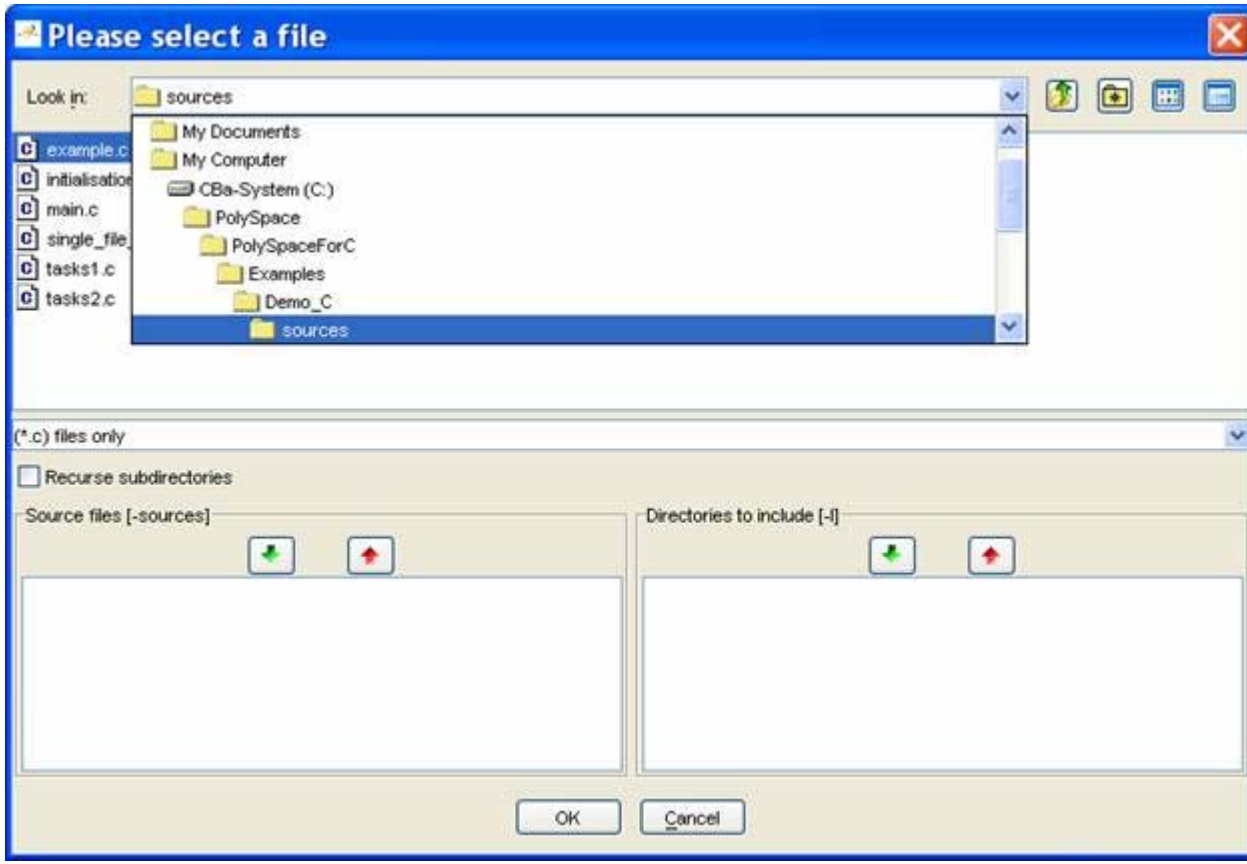


4 Start by updating the result directory name by clicking on the browse button  .

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

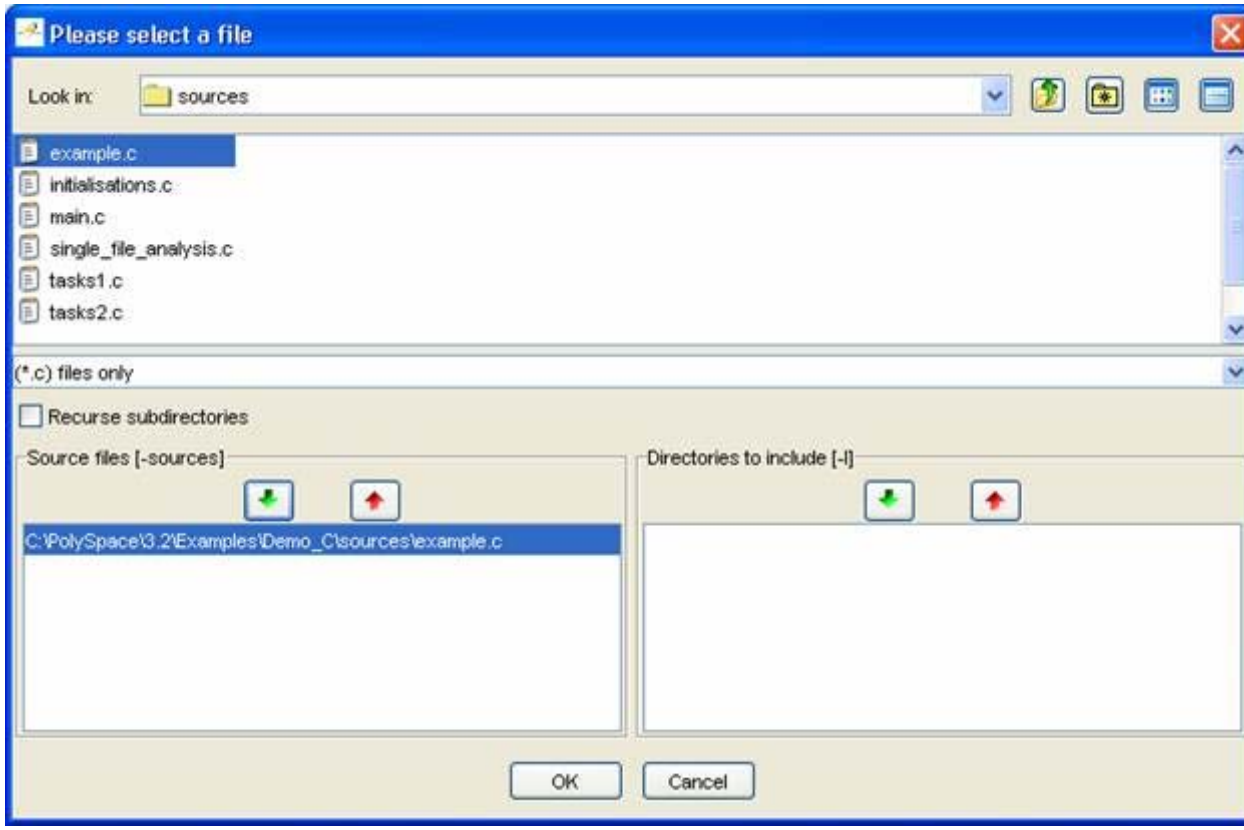
5 Now, click on the  button (right of the “New Project” label).

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

7 Select `example.c` 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 C - New_Project” window.

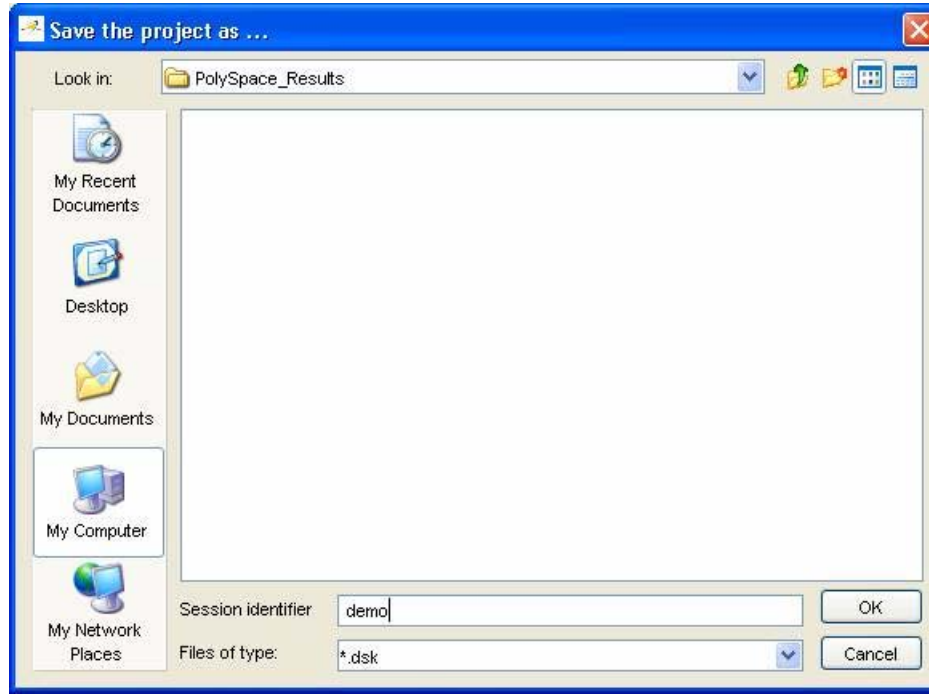
Note It is also possible to drag a directory or source files and drop them directly in the “File Name/Absolute Path” part (top left of PolySpace Client) without using the “Please select a file” window.

PolySpace™ Client™ for C/C++: Running the Analysis

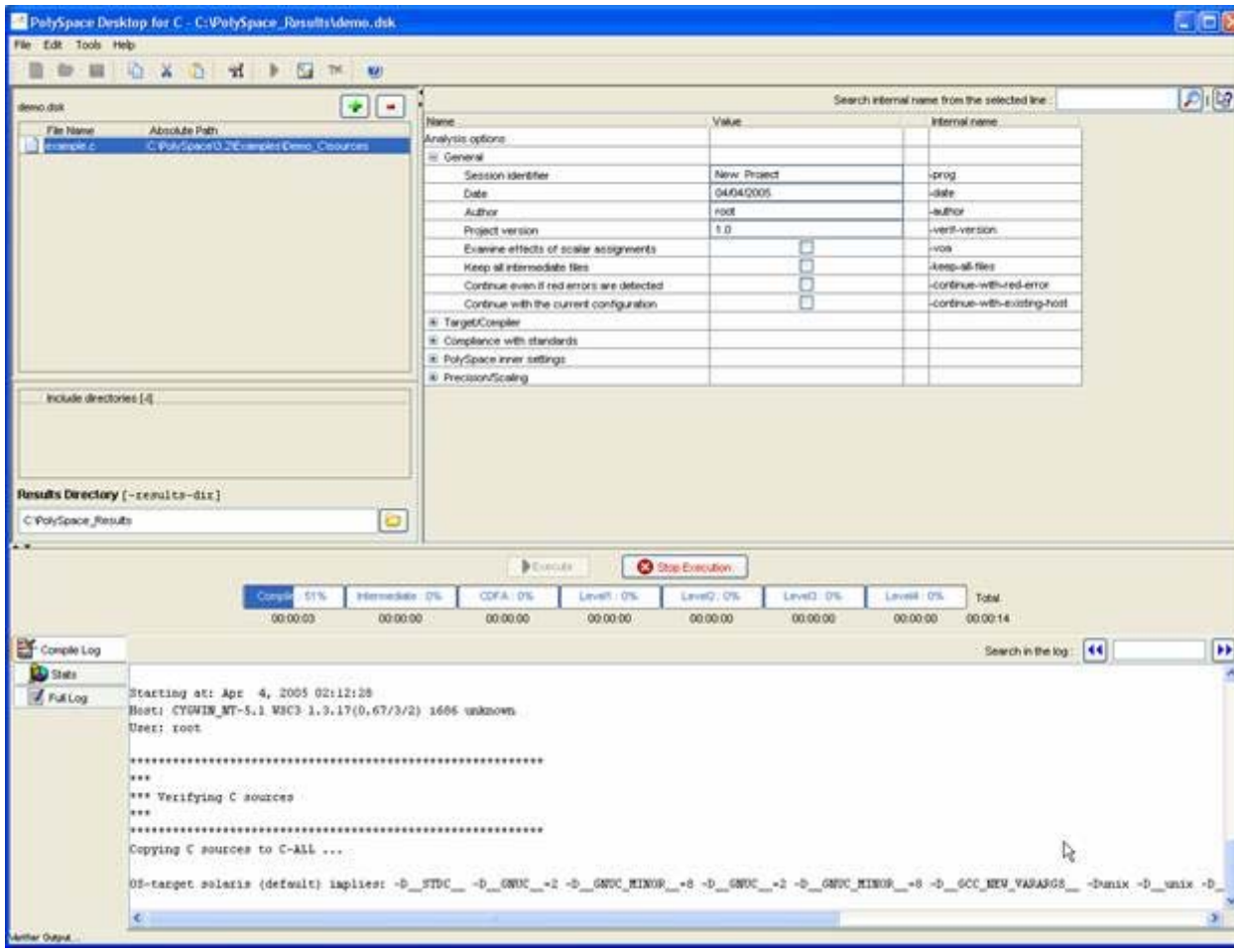
To run the analysis:

1. Click on Execute to start the analysis. Alternatively, you can click on the button in the title bar to run PolySpace™ Client™ for C/C++ analysis 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 `demo` and save it under PolySpace™ result directory. The full name of that file will be `demo.dsk`.



2. Click on OK to go back to the “PolySpace Desktop for C - New_Project” window and click again on Execute to proceed.

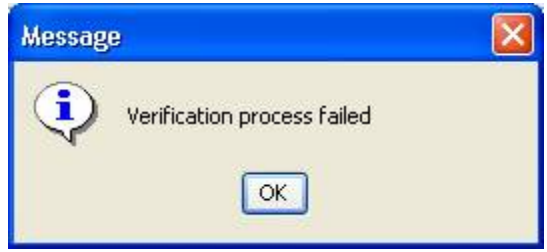


A progress report is displayed in the bottom part of the graphical interface, indicating that the analysis is being performed. The Execute button is also grayed out.

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

Parsing Errors During Preliminary Analysis Stages

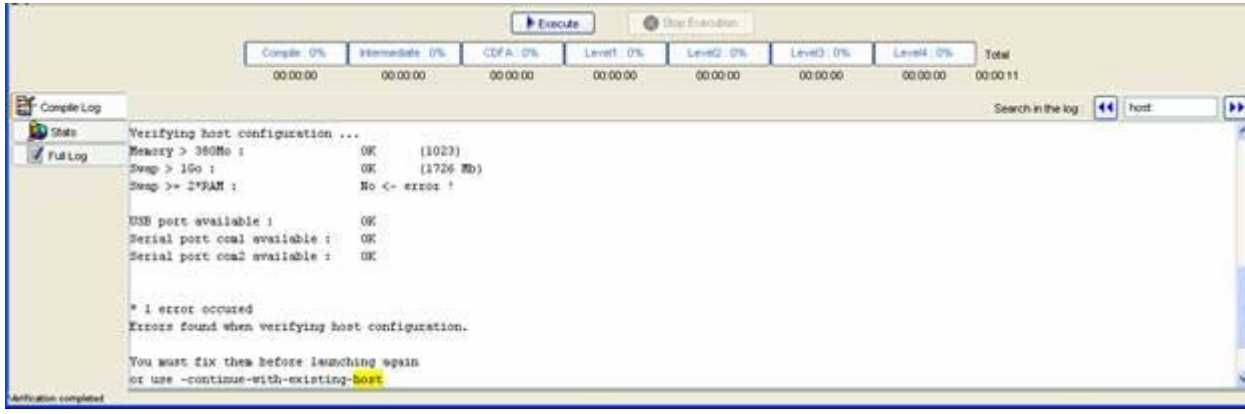
After some checks, the software will show an error message:



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.

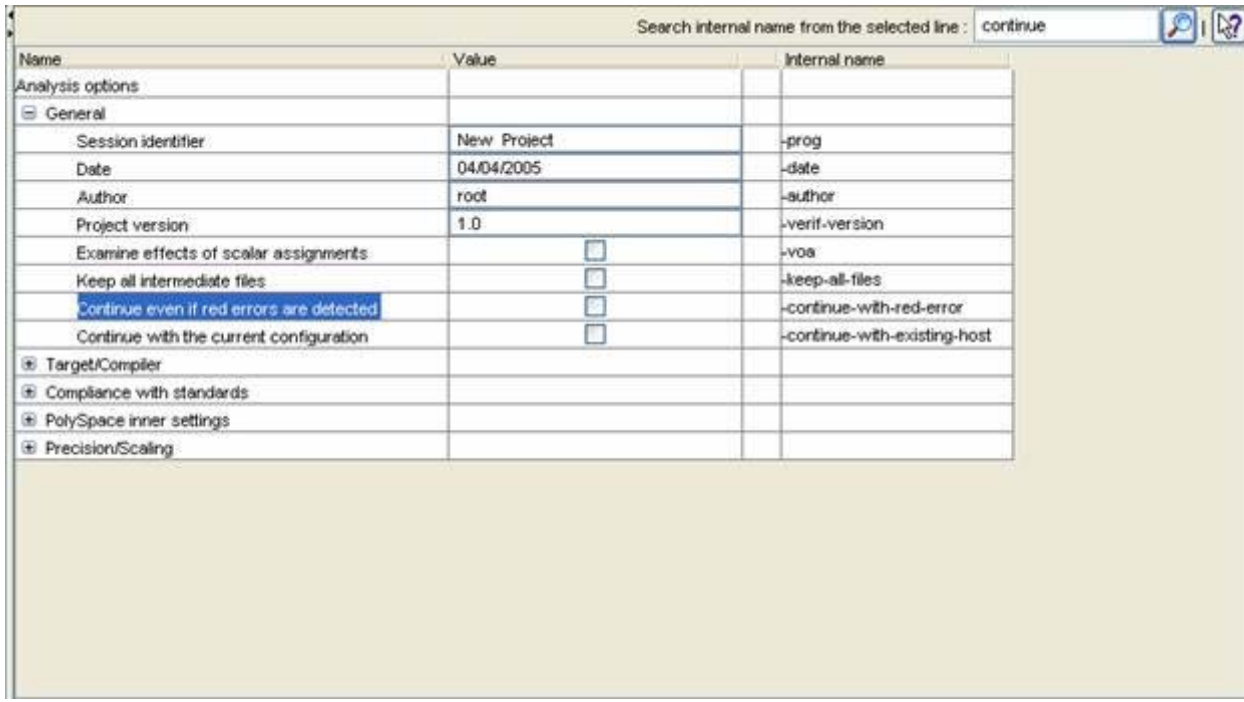
If the error message corresponds to the one shown above and in order to continue analysis, you can either:

- 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).

3 Click  .

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



4 Check the box

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

It is also recommended to select the -continue-with-red-error option. Indeed, `example.c` contains - on purpose - code with some definite errors, later called red errors. This option allows you to continue the analysis even if red errors are detected in previous passes.

Continue even if red errors are detected	☑	-continue-with-red-error
Continue with the current configuration	☑	-continue-with-existing-host

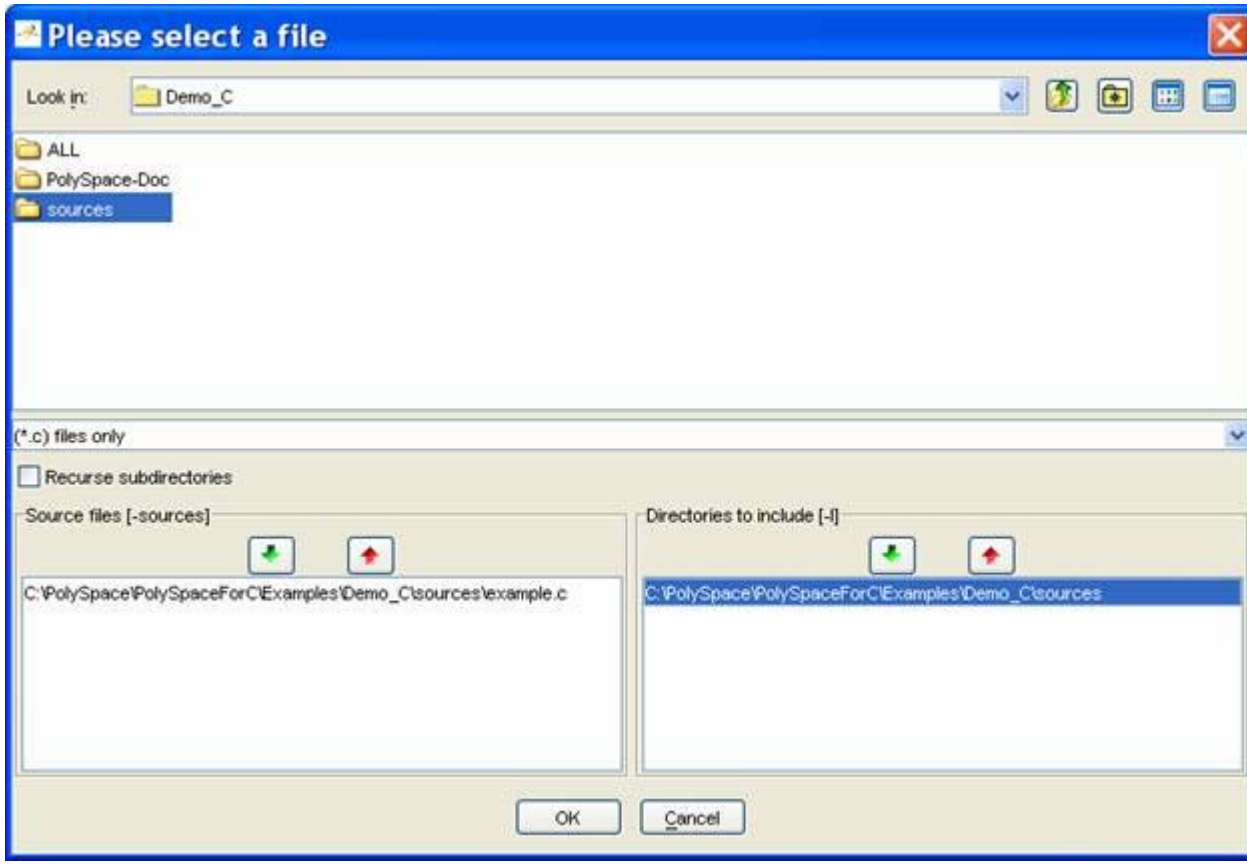
Second Possible Cause for the Error Message: Information About Header Files

Another cause of error may be that PolySpace Desktop is missing 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 products are not associated with one particular compiler, it is mandatory to indicate where library files are stored.

In our `example.c` file analysis, the related `math.h` file is located in the same directory as the C file: *`PolySpaceInstallDir`*`\Examples\Demo_C`.

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



2 Select *`PolySpaceInstallDir`*`\Examples\Demo_C\sources`, where `math.h` is located.

3 Click on  in the “Directories to include [-I]” section, then click **OK** to close the window.

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

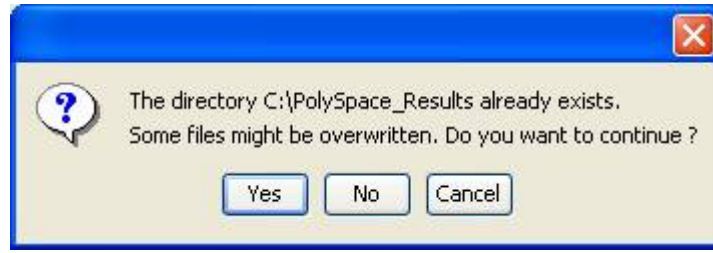
It is also possible to drag a directory and drop it directly in the "include directories [-I]" part (top left of PolySpace Client) without using the "Please select a file" window.

Progression of the Analysis

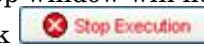
1 Click on 

to restart the analysis. 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:

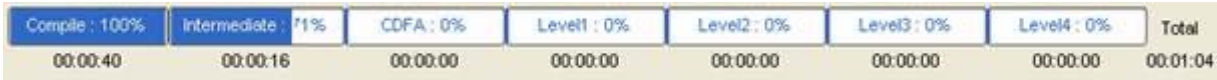


2 If this happens, click **Yes**.

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.

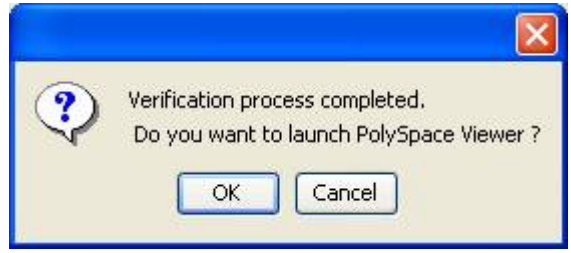
4 Click 

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, the software 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. , and if no other analyses are running, you can access the results via the icon in title bar.

3

PolySpace™ Viewer — Exploring Results

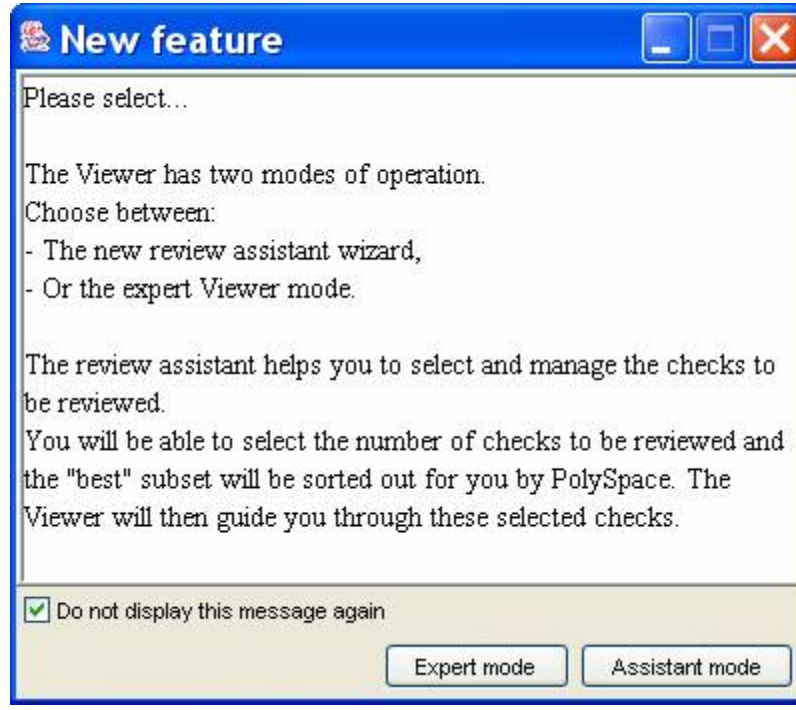
Overview

This step illustrates how to explore analysis results that were generated by either PolySpace™ Client™ for C/C++ or PolySpace™ Server™ for C/C++ products. We review the results of the analysis of `example.c` performed in Chapter 2, “PolySpace™ Client — Analyzing a Single C File”.

If you clicked **OK** at the end of the previous analysis (see previous section), 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 software. It concerns the “best” subset

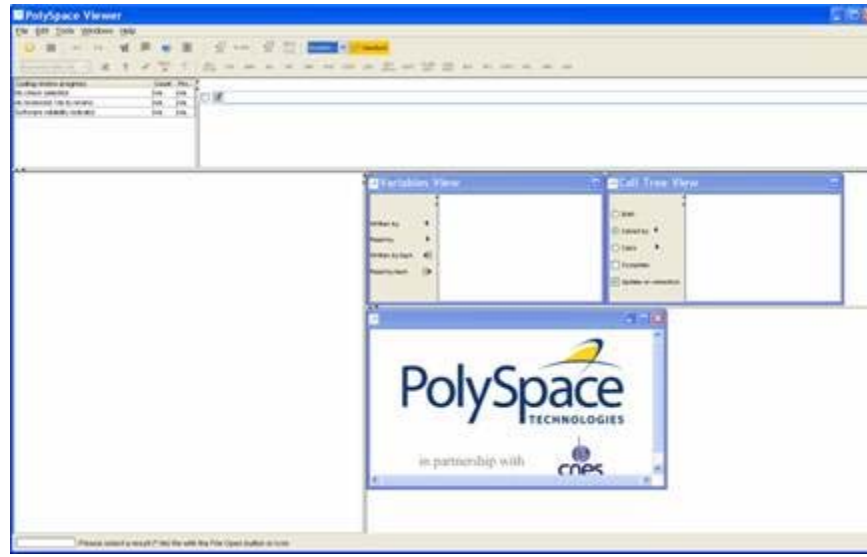
of checks sorted out for user. The PolySpace Viewer will then guide user through these selected checks.

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:



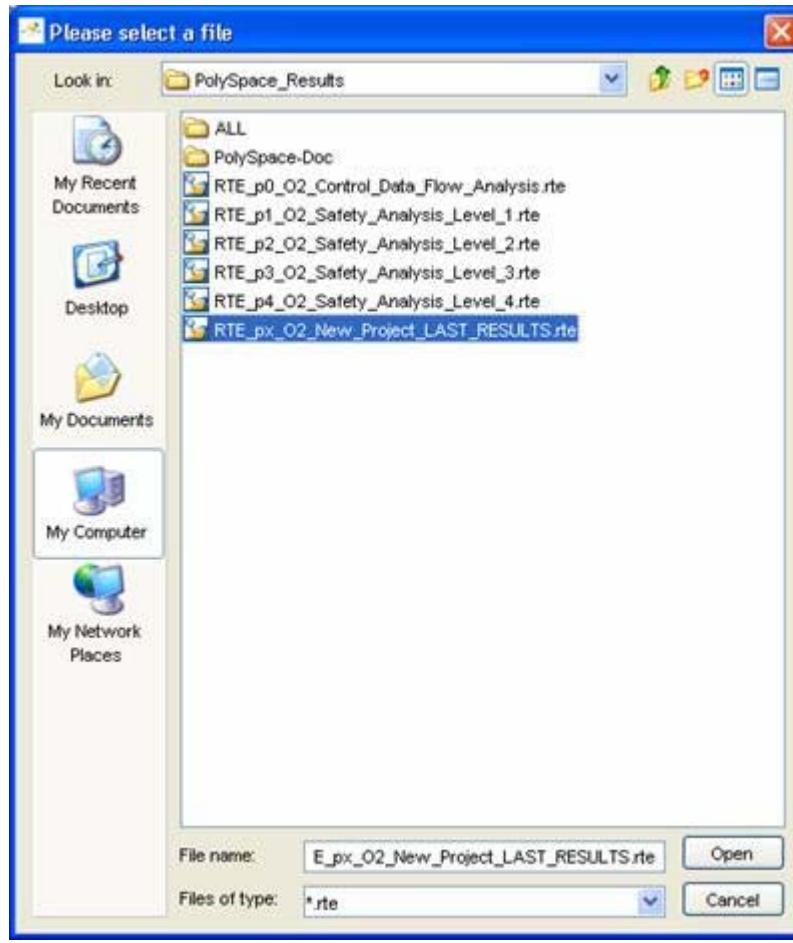
1 Click **File > Open** to load result files.

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

PolySpaceInstallDir`\Examples\Demo_C\RTE_px_O2_Demo_C_LAST_RESULTS.rte`

---

2 Select the following file located in `C:\PolySpace_Results`.

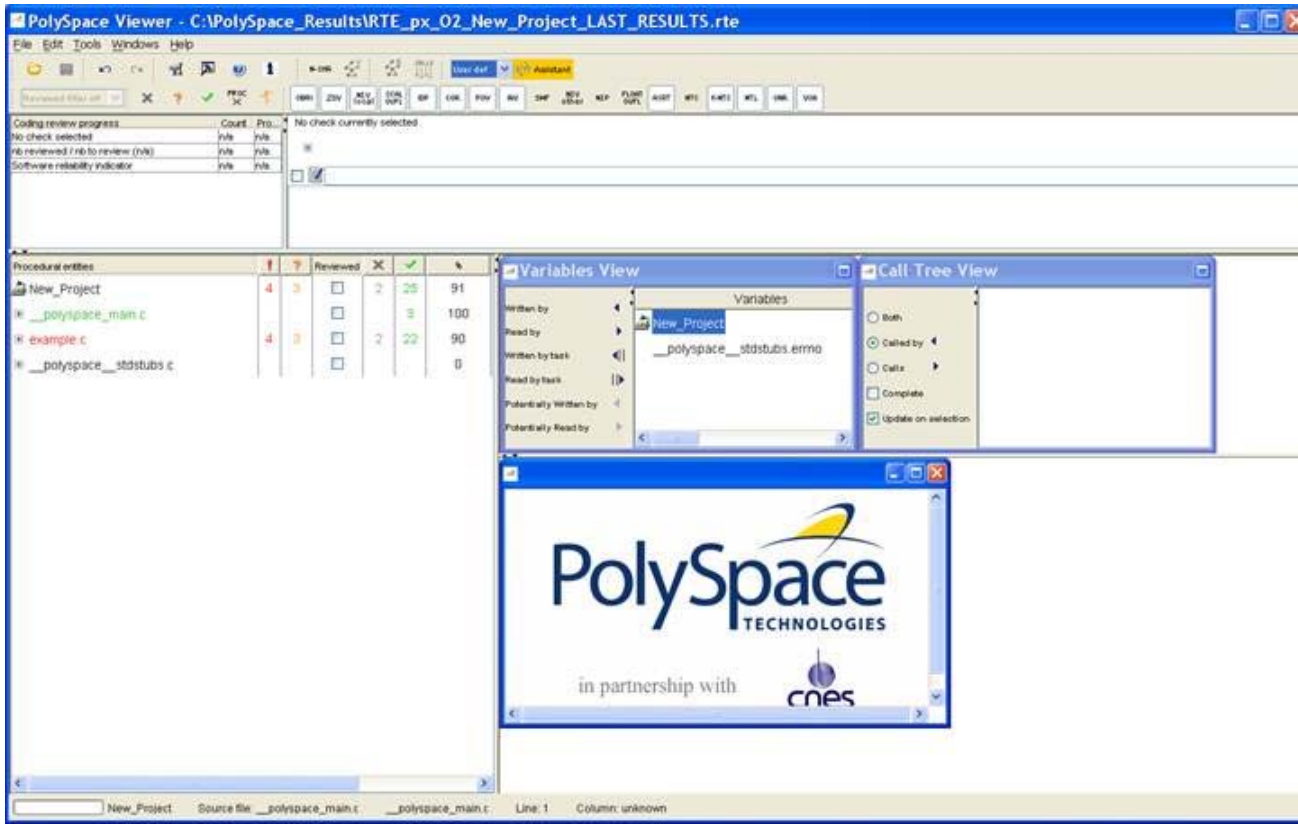


3 Click **Open** to proceed with further steps

Note The `RTE_px_O2_Demo_C_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_Level4.rte` file. Lower level files represent lower precision analysis.

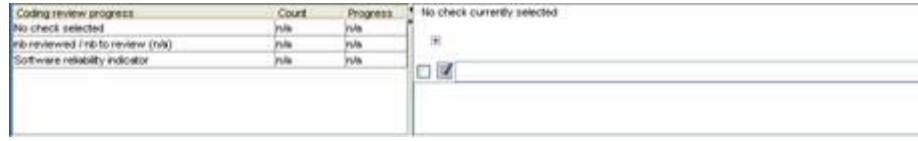
Reviewing PolySpace™ Results in "Expert" Mode (example.c)

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 occur 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)

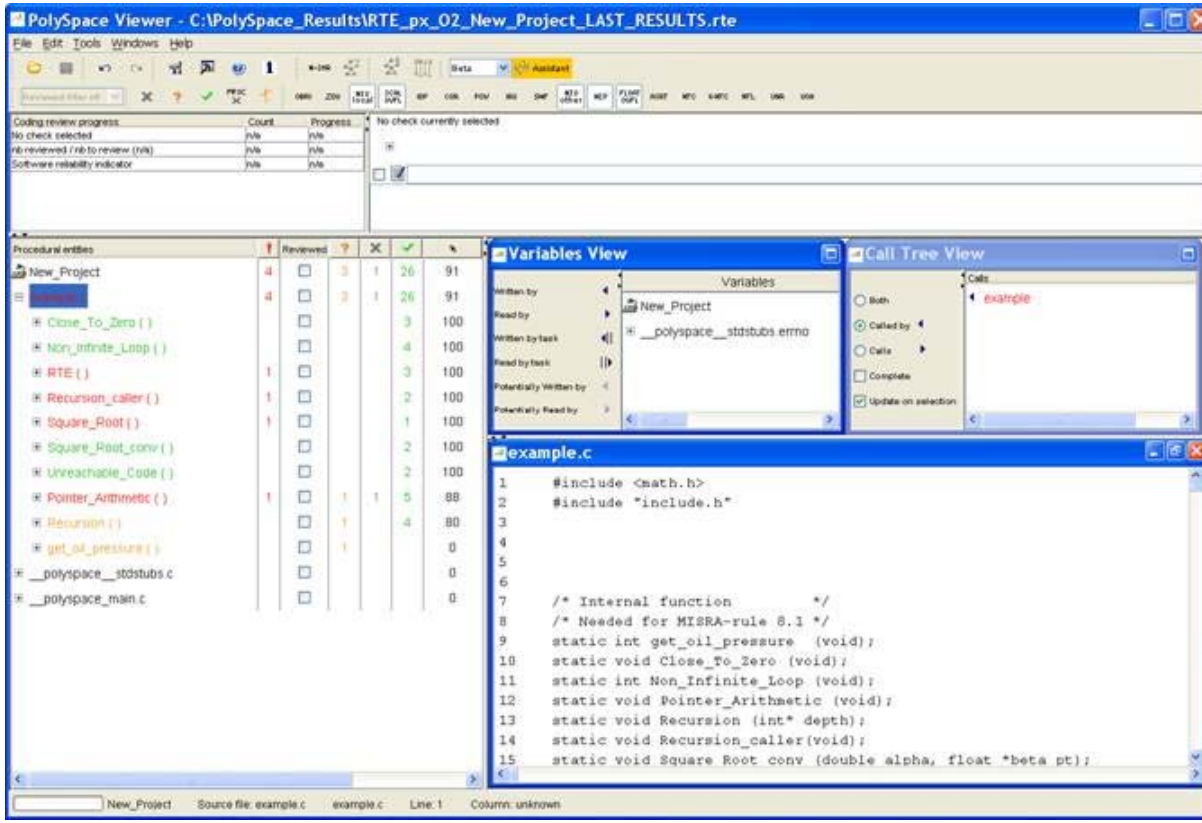
Each file and underlying functions in the procedural entities view (or RTE view) is colorized according to the most critical error found:

- `__polyspace_main.c` — This file contains the main which was automatically generated. All checks there are green: no run-time error (or RTE) has been found.
- `example.c` — This file is red: one or more *definite* run-time errors have been found in it.

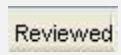
- `__polyspace_stdstubs.c` — contains no checks. It contains stubs of standard functions part of libclibrary used in `example.c`.

Click once on the ⊞ left of `RUNTIME_ERROR` to find out more about this package.

RUNTIME_ERROR is expanded and the list of functions defined within RUNTIME_ERROR is displayed. The functions in red or grey have code sections that need to be inspected (PROCEDURE_ZDV, SQUARE_ROOT, etc.) first because they are definite diagnosis of PolySpace verification (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.
	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.
	Allows marking reviewed checks.

Let's have a look at some errors found by PolySpace verification in `example.c`.

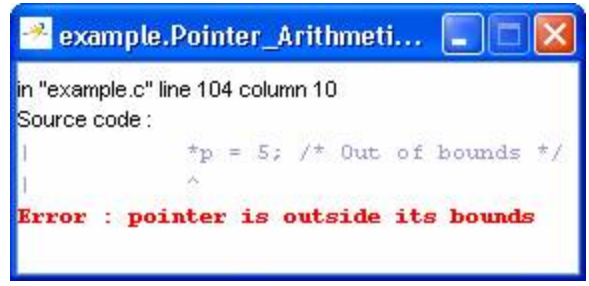
First Example of Runtime Error: Memory Corruption

To investigate the first error:

1. Click on ⊞ to expand "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 verification checked within the "Pointer_Arithmetic()" function.

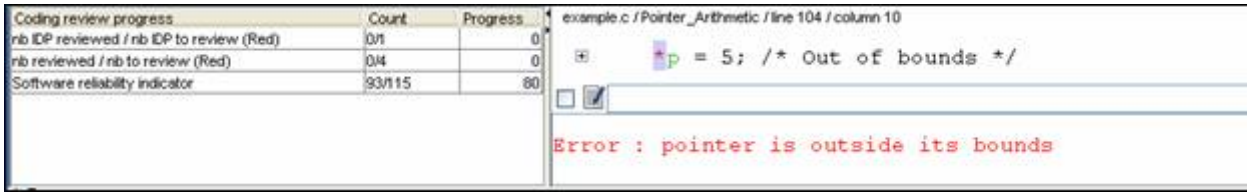
2 Click on the red "IDP.9" item - which stands for Illegal De-referenced Pointer -, to precisely locate this error in the source code. The bottom right section is updated showing the location of the "IDP.9" item.

3 Click on red symbol in the source code at line 104. An error message is opened:

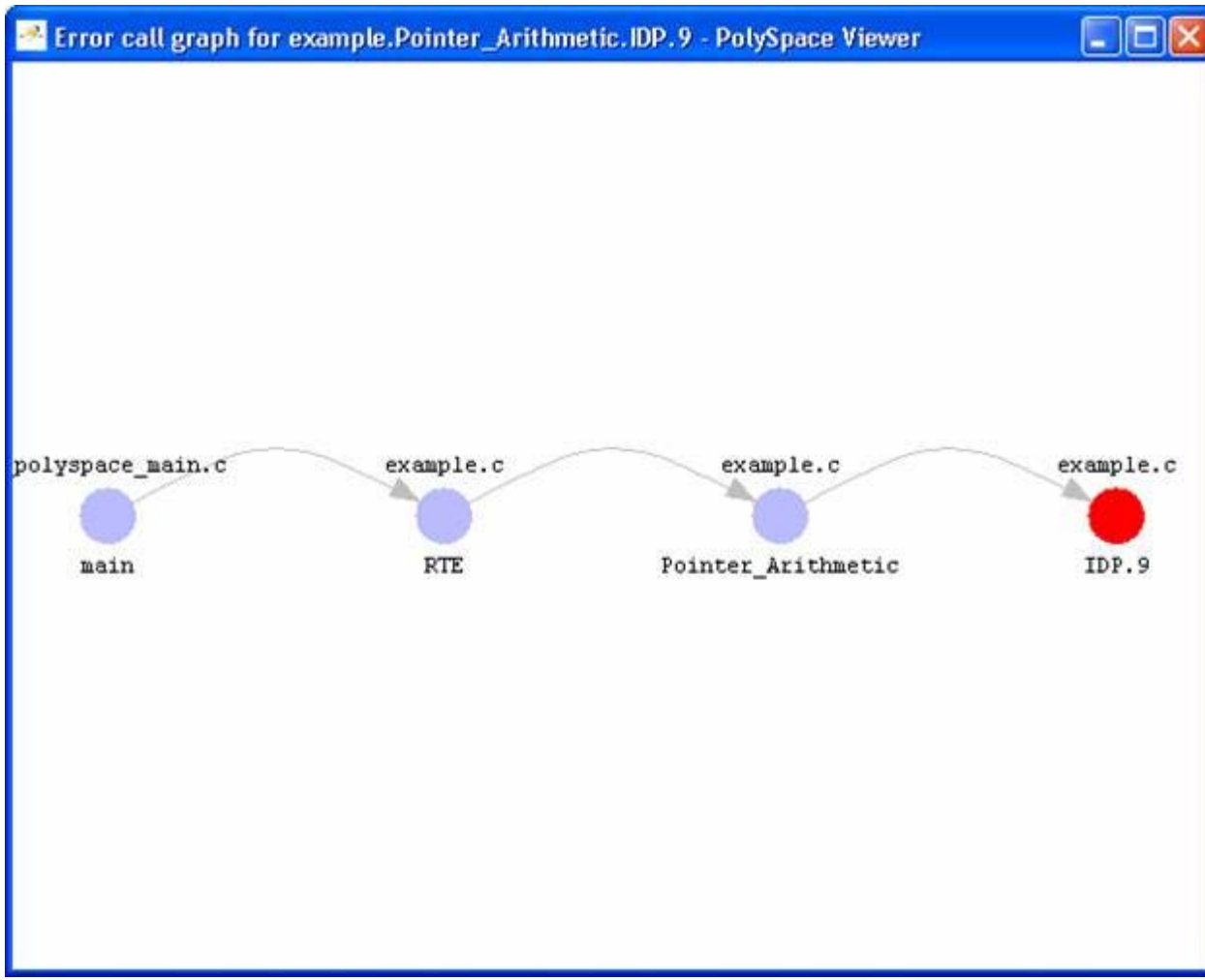


Pointer p is de-referenced outside of its bounds. Indeed, at the line 71 the instruction "*p = 5;" corrupts the memory as it puts the value "5" outside of the array "tab" pointed to by the pointer "p".

Information about this red IDP is also accessible in the right windows below the toolbar line and the left one gives some statistic about all the IDP in the analysis:



4 You can also see the calling sequence leading to that particular red code section. To do so, select "IDP.9" 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:



Second Example of Runtime Error: Unreachable Code

Select "Unreachable_Code()" in the RTE View. You can see that "x = x + 1" is unreachable (gray color on each check) because of the non satisfied boolean condition: "x" is never negative when evaluating "x<0". PolySpace verification has detected some dead code.

```
199   static void Unreachable_Code(void)
200         /* Here we demonstrate PolySpace Verifier's ability to
201            identify unreachable sections of code due to the
202            value constraints placed on the variables.
203         */
204   {    int x = random_int();
205    int y = random_int();
206
207    if (x > y)
208      {
209        x = x - y;
210        if (x < 0)
211          {
212            x = x + 1;
213          }
214      }
215
216    x = y;
217   }
```

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 verification 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 verification 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 the PolySpace Viewer.

In a First Example of the Second Set: Arithmetic Error

Click ⊞ to expand "SQUARE_ROOT" function. You can see the source code view in the bottom right.

You can also display the call tree for that function by using the "Windows" menu (see previous paragraph).

"Square_Root()" is called by RTE function from "example.c". It is displayed as "example.RTE" in the "*Call tree view*" window (right of the top right section).

"Square_Root" calls "random_float" (automatically stubbed function), "Square_Root_conv" (from example.c) and "sqrt" (standard library).

```
179   static void Square_Root_conv (double alpha, float *beta_pt)
180         /* Perform arithmetic conversion of alpha to beta */
181   {
182     *beta_pt = (float)((1.5 + cos(alpha))/5.0);
183   }
184
185   static void Square_Root (void)
186   {
187     double alpha = random_float();
188     float beta;
189     float gamma;
190
191     Square_Root_conv (alpha, &beta);
192
193     gamma = (float)sqrt(beta - 0.75);   /* always sqrt(negative number) */
194   }
```

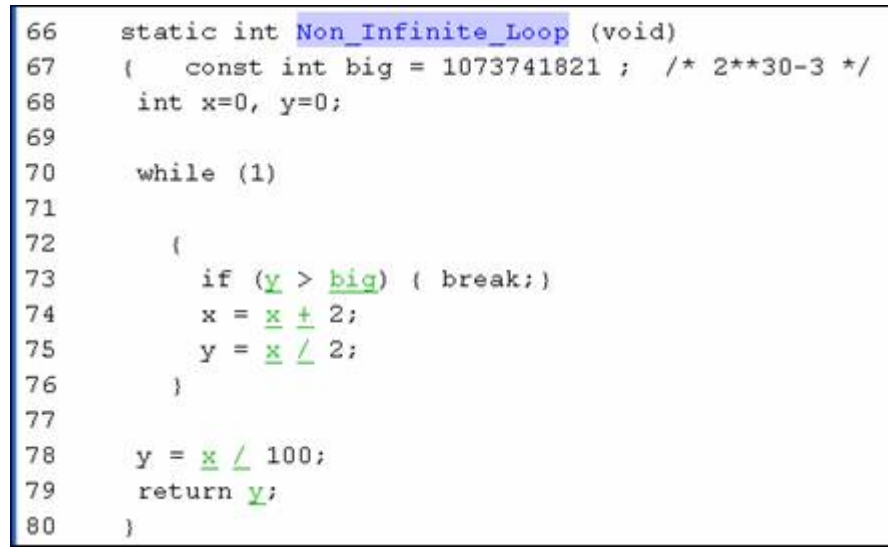
The green sections into the source code view are error-free but the red (**sqrt**) is an issue that needs to be fixed. Indeed, when the local float variable gamma is computed in the line “gamma=sqrt(beta - 0.75);“, the operation will cause a run-time error, as the parameter passed to “sqrt” is always negative.

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

---

Second Example of the Second Set: Non-Infinite Loop

Select “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 verification confirms that the function always terminates.



```
66    static int Non_Infinite_Loop (void)
67    {   const int big = 1073741821 ;  /* 2**30-3 */
68     int x=0, y=0;
69
70     while (1)
71
72       {
73         if (y > big) { break;}
74         x = x + 2;
75         y = x / 2;
76       }
77
78     y = x / 100;
79     return y;
80    }
```

Third Example of the Second Set: Non-Infinite Loop

Select "Recursion_caller()": The first call to Recursion is in red because when a negative parameter is passed, Recursion makes a division by zero (See the "Recursion" function). PolySpace verification also checks recursive constructs:

```
137   static void Recursion (int* depth)
138        /* if depth<0, recursion will lead to division by zero */
139   {    float advance;
140
141    *depth = *depth + 1;
142    advance = 1.0f/(float)(*depth);  /* potential division by zero */
143
144
145    if (*depth < 50)
146      {
147        Recursion(depth);
148      }
149   }
150
151   static void Recursion_caller(void)
152   {    int x=random_int();
153
154
155    if ((x>-4) && (x < -1))
156      {
157        Recursion( &x );   // always encounters a division by zero
158      }
159
160
161    x = 10;
162    if (random_int() > 0)
163      {
164        Recursion( &x );   /* never encounters a division by zero */
165      }
166   }
```

Advanced Results Exploration

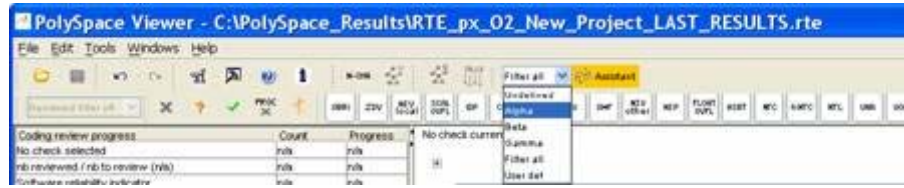
You can filter the information provided by PolySpace software to focus on the type of errors you wish to investigate.

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 Square Root function that we have examined in the previous section.

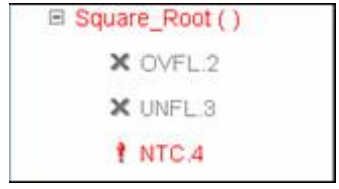
Gamma Mode

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 "SQUARE_ROOT".



This list of acronyms - for type of operations checked - shows what PolySpace verification automatically analyzed for you.

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 Beta.

2 Select “Pointer_Arithmetic()” in the “Procedural entities”.

3 Click ⊞ to get the list of the checks.

Alpha Mode

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

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:

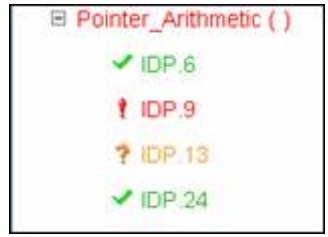
Note When the mouse pointer moves on the filter, a tool tip gives its definition.

- Click 

(top of the window) to suppress all checks, then click 

.

  You will get list of checks containing only IDP (**I**llegal **D**ereference **P**ointers) 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:

Miscellaneous

The  icon gives access to the PolySpace documentation. 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

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

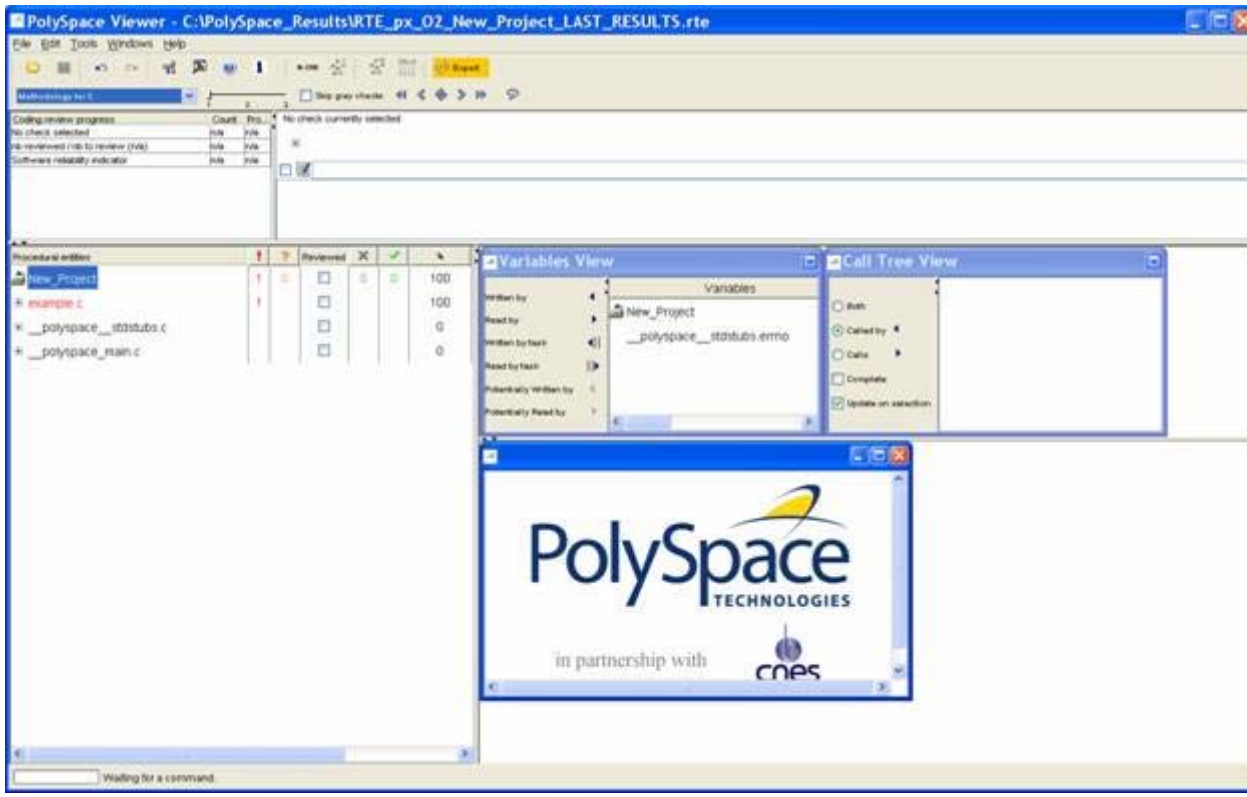
- Do all checks need to be reviewed?
- What are the checks to review?
- How many?
- What is the best order?

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.

To open the assistant:

1. If the PolySpace Viewer is still open, close it by clicking on the upper right  symbol.
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:



Assistant Dashboard

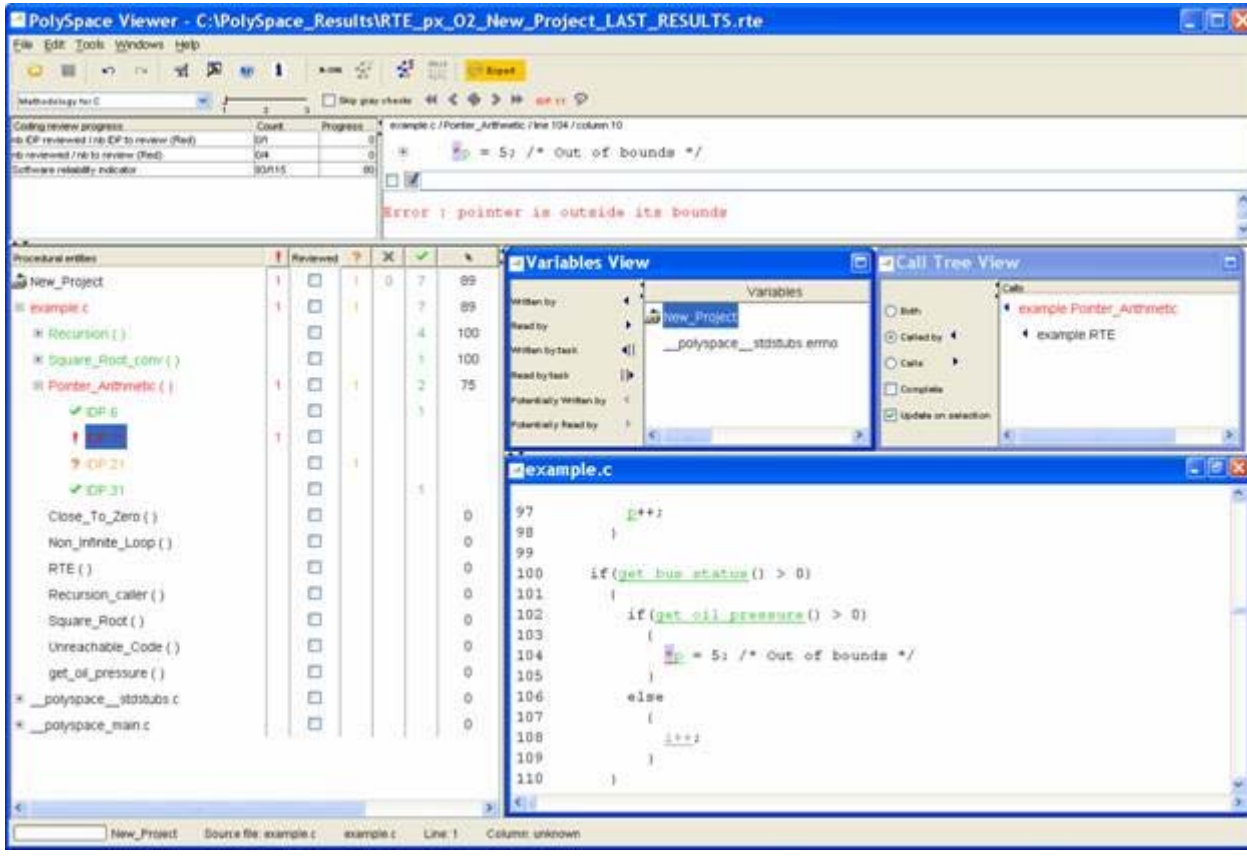
The second line of buttons on the toolbar and the two views just below are the navigation centre based on the methodological method used in the assistant mode:

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 has been refreshed with the first check selected by the Methodology of review:

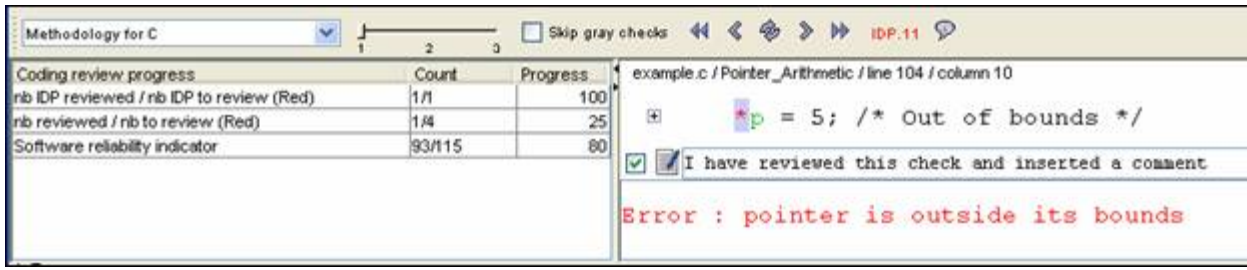


The Methodological dashboard gives details and allows reviewing the check. On the selected check, it is possible to mark the fact that it has been reviewed.

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.

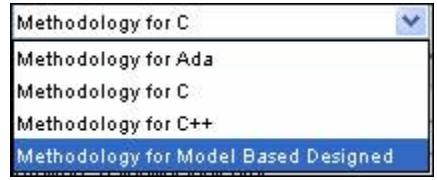
Other buttons in the Methodological dash board allow navigating to previous check, coming back to current one

and going to next

/ previous

category of reviewed checks selected by the Methodology.

Choose a Methodological Assistant



and associated levels 

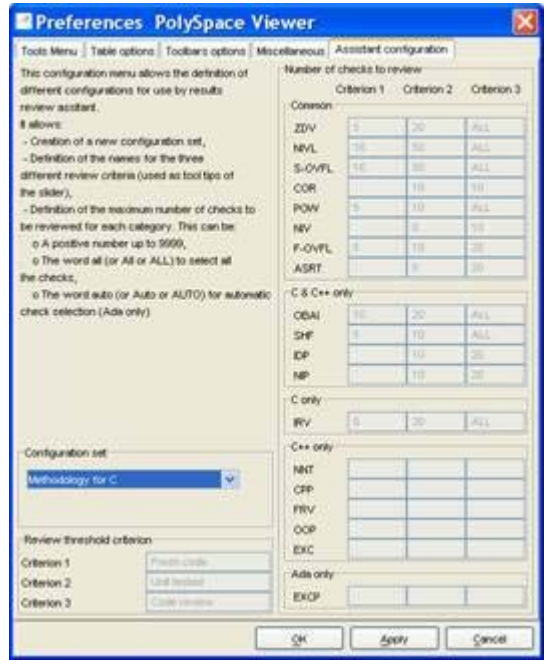
have been pre-selected by PolySpace software.

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.

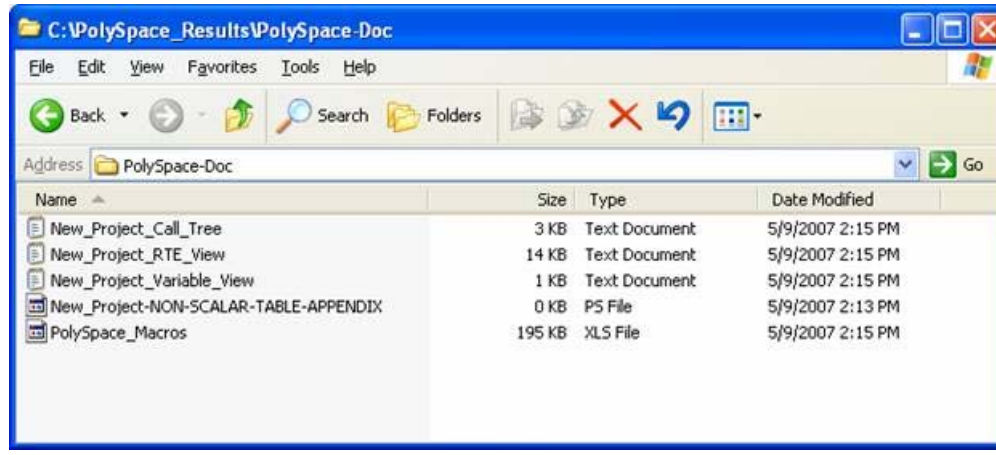
Note You cannot change an existing configuration except by duplication and refinement.

Report Generation

When PolySpace™ software performs an analysis, it generates textual files that can be used to generate Microsoft® Excel® reports. These files are located in the results directory (see `C:\PolySpace_Results\PolySpace-Doc` or *`PolySpaceInstallDir`*`\Examples\Demo_C\PolySpace-Doc`).

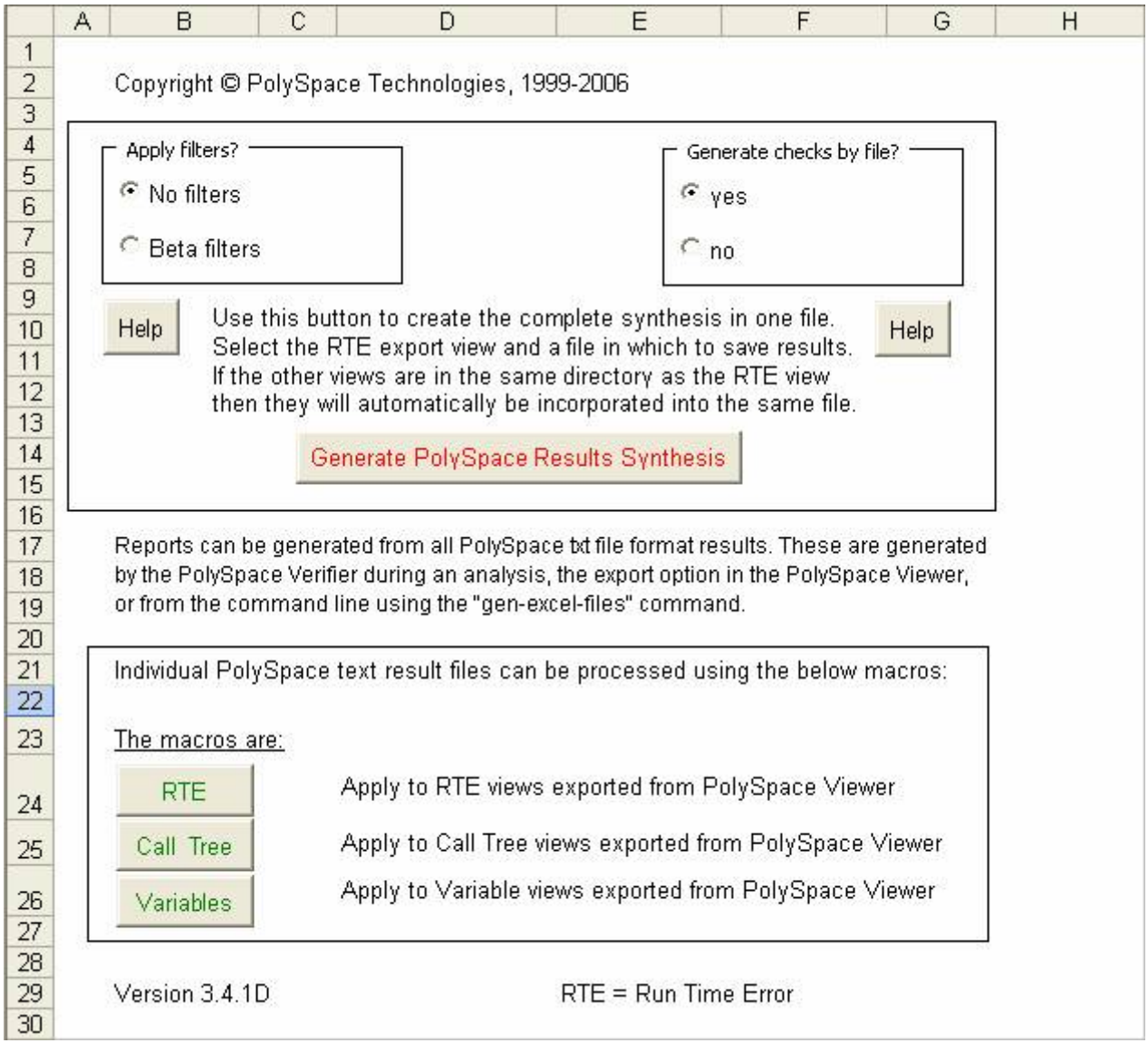
All views (except source code) are printable and can be exported to textual or Excel format (protected by license).

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



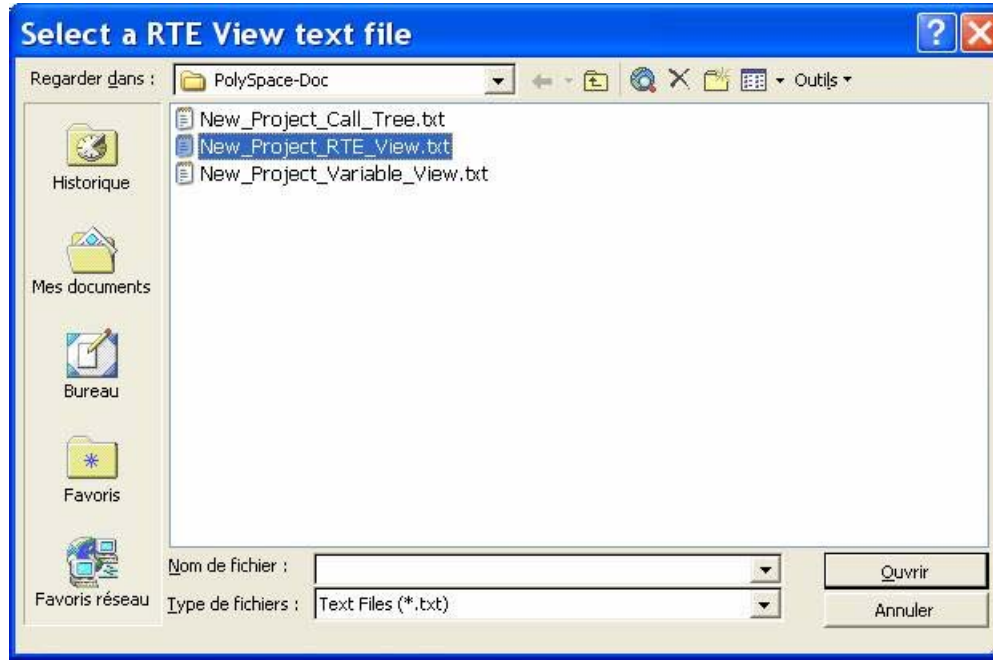
To generate a report:

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



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.

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	RTE Statistics						
2	Check category	Check detail	R	O	Gy	Gr	% proved
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 the results review.

4

Setting Up and Launching the MISRA-C Checker

Before You Begin (p. 4-2)	Describes how to activate the MISRA C checker
Selecting MISRA C® Rules to Check (p. 4-5)	Describes how to configure rules for the MISRA® checker
Running the MISRA® Checker (p. 4-12)	Describes how to run an analysis with the MISRA checker

Before You Begin

In this section...

Overview

This chapter describes the basic steps to add the MISRA C® Checker in the analysis of `example.c`. This operation takes place during ANSI® C compliance phase of the analysis.

Activating the MISRA C® Checker

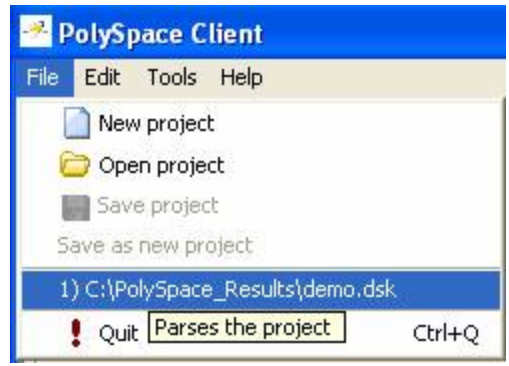
The project created during first step of this guide needs to be updated by activating the “**Check MISRA rules**” option while selecting the parameters.

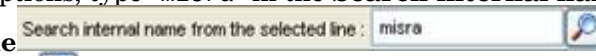
Note If the PolySpace™ Client™ for C/C++ product is already opened with the project defined previously (refer to “Setting Up a PolySpace™ Client™ for C/C++ Analysis” on page 2-4), you can skip the first two steps.

---

1 If the PolySpace Client for C/C++ window has been closed, please open it again by double-clicking on the Client icon:

2 Select the saved project:

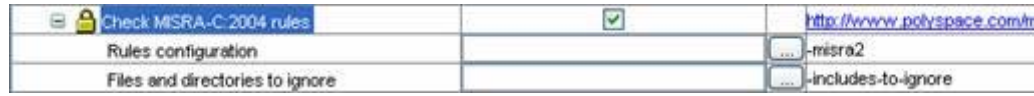


3 To set up the -misra2 options, type "`misra`" in the **Search internal name from the selected line** 

(top right) box, then click  .

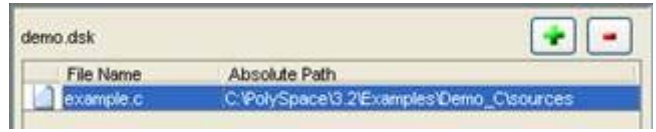
The software shows all options containing `misra` in the set of options part below.

4 Select the **Check MISRA-C: 2004 rules** option and expand it to see the two associated options `-misra2` and `-includes-to-ignore`:

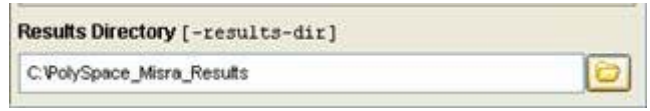


We will detail these two options below.

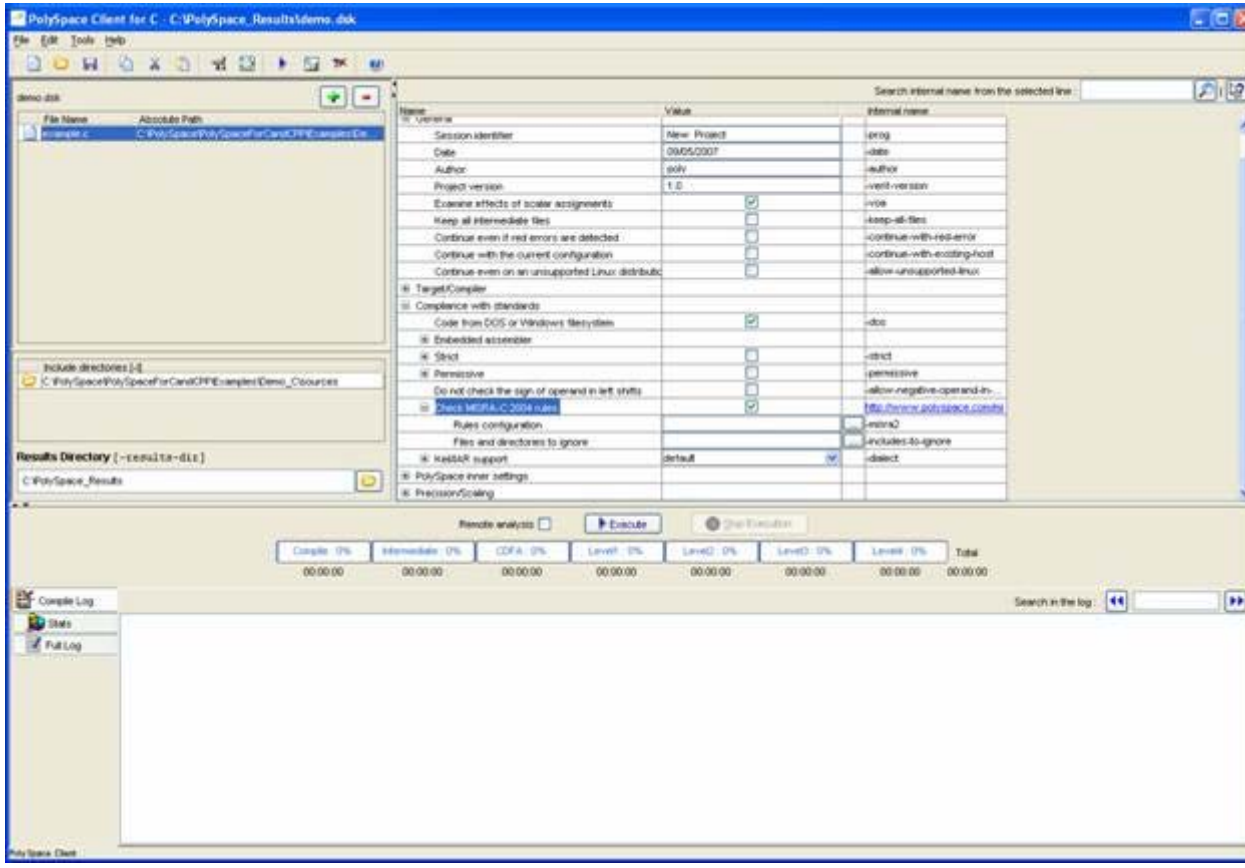
5 Make sure `example.c` is selected (location from *`PolySpaceInstallDir`*`\Examples\Demo_C\sources`):



6 Update the results directory:



The PolySpace™ Launcher should now look like this:



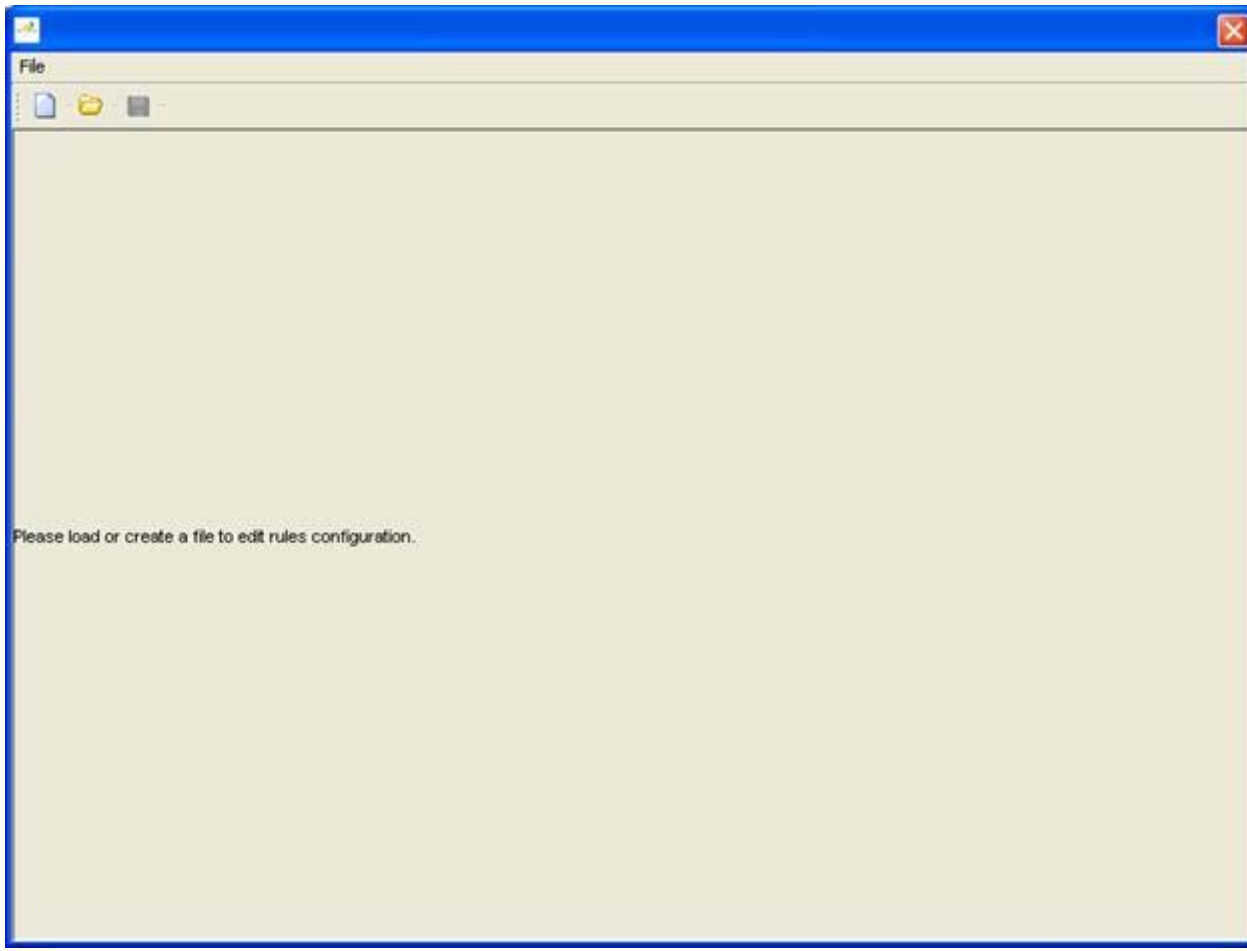
Selecting MISRA C® Rules to Check

In this section...

File Configuration

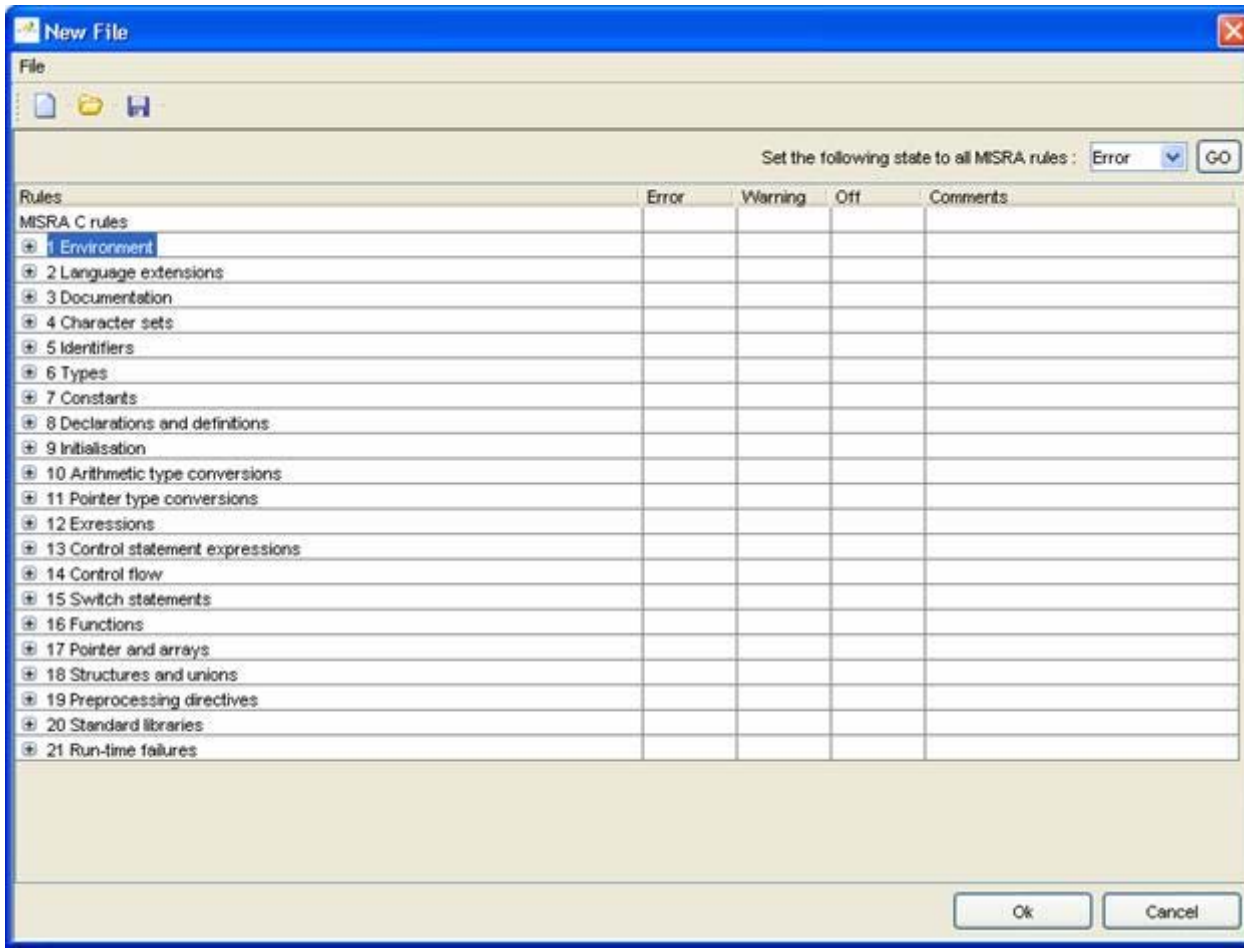
1 Click on ... to invoke "`Rules configuration`."

Note This button was enabled during activation of "`Check MISRA-C: 2004 rules`."



2 Click to create a new MISRA C® configuration file.

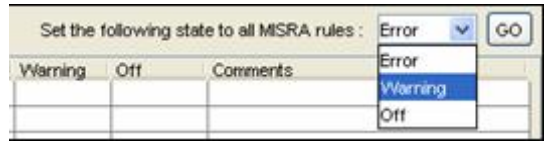
The previous window is updated.



3 Set each rule as follows:

- `Error` — this MISRA C rule must be respected. If one or several errors are detected, the analysis will stop at the end of the compilation phase.
- `Warning` — if this MISRA C rule is not respected, a warning will be displayed, but the analysis will continue.
- `Off` — the MISRA C rule will not be verified by PolySpace™ MISRA® Checker module

Note The default setting for all rules is `Warning`.



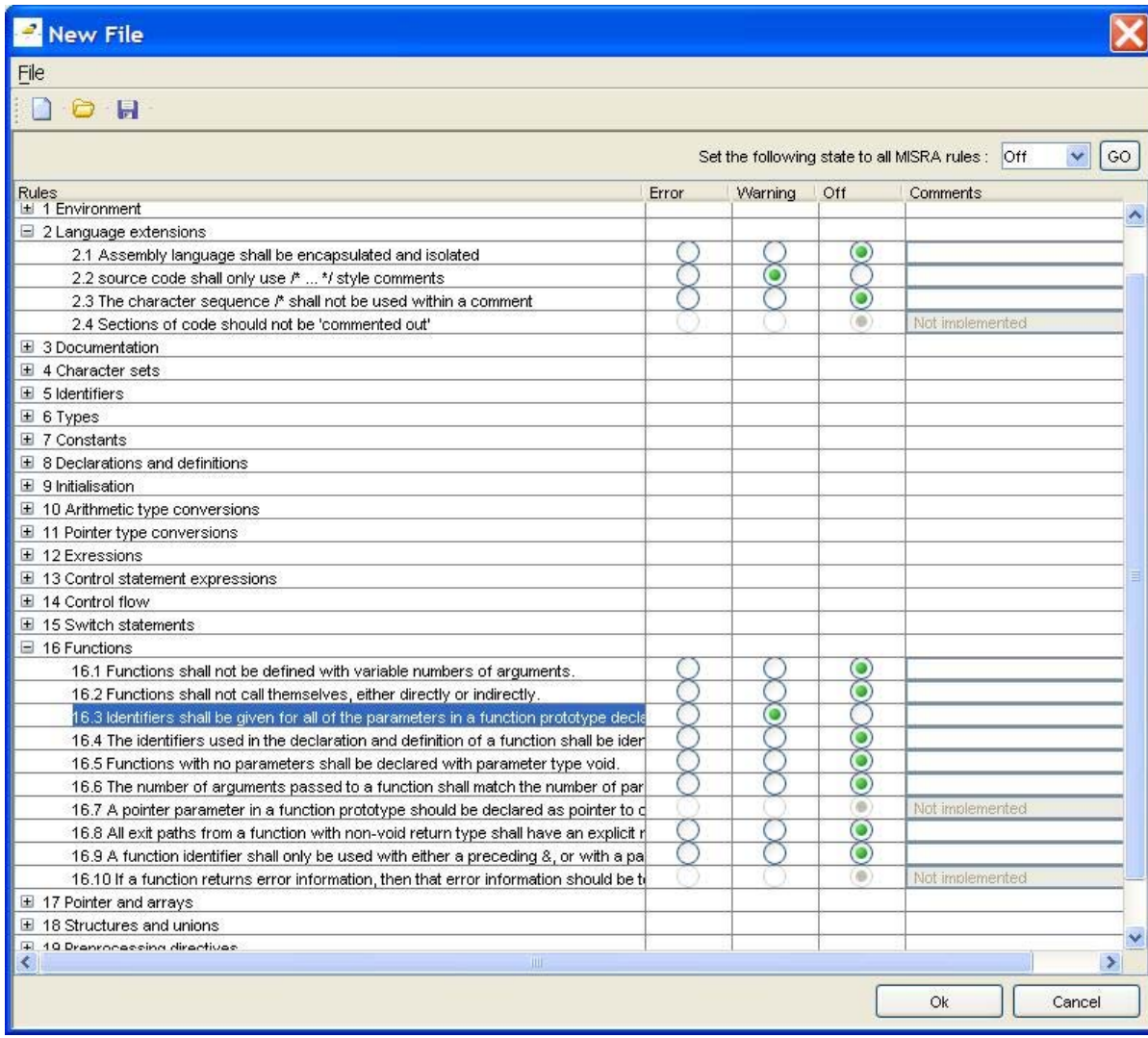
4 For the MISRA C check of `example.c` file, please update the setting to `Off` for all rules and apply it using the GO button

5 Click on ⊕ to expand the set of rules **16. - "Functions".**

The status of some of the underlying rules cannot be modified (rules **16.7** and **16.10**), others are "`Off`".

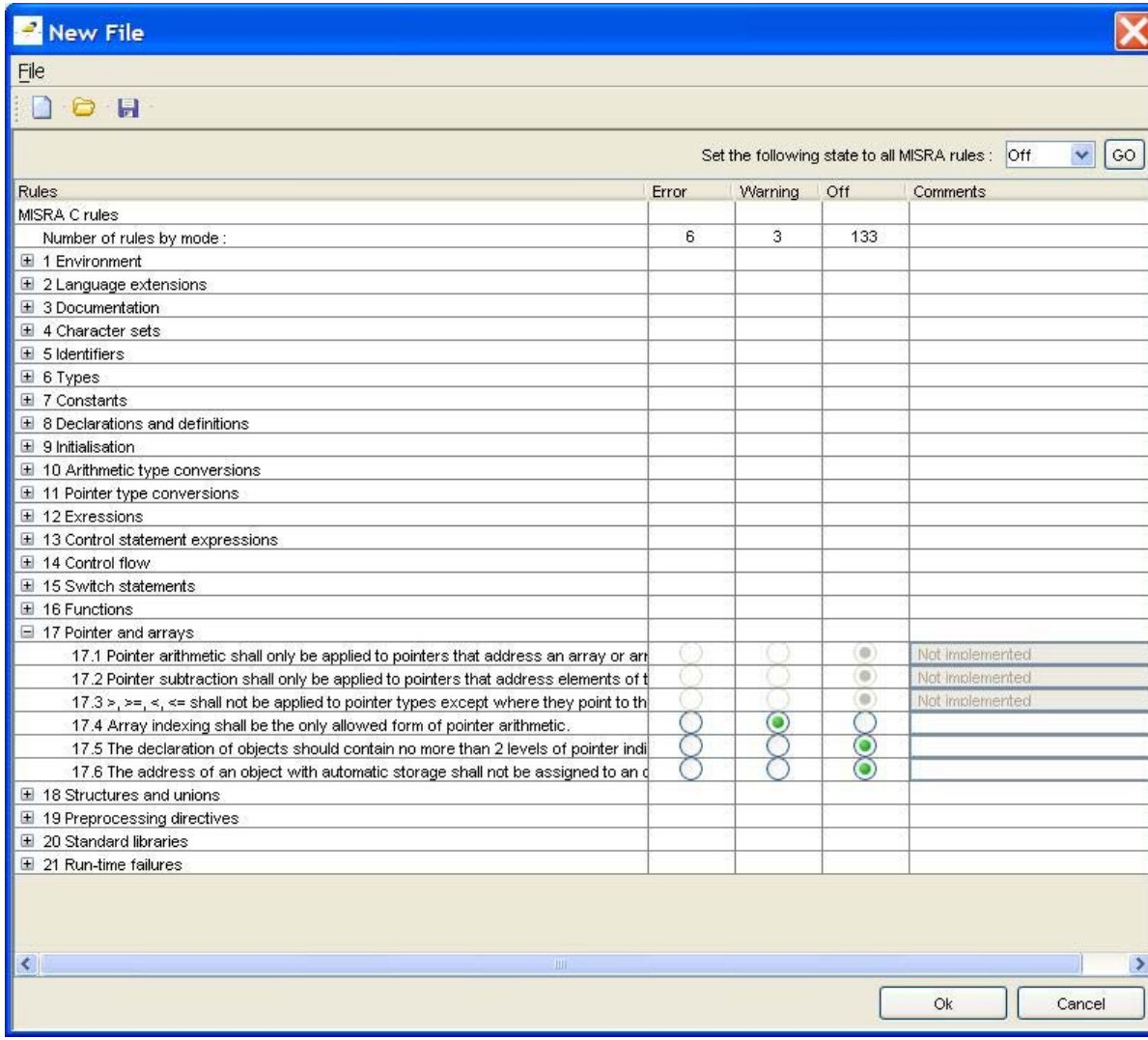
6 Click on the **Error** column for rule **16.3.**

The green dot ⊙ moves from column **Off** to column **Error**. This means that PolySpace MISRA Checker will verify whether the rule 16.3 ("Identifiers shall be given for all of the parameters in a function prototype") is respected and will stop after the ANSI® compliance checking phase if this is not the case.



7 Click on ⊞ to expand the set of rules **17**- “Pointers and arrays,” then select **Warning** for rule **17.4.**

This means that PolySpace MISRA Checker will verify whether the rule 17.4 (“Array indexing shall be the only allowed form of pointer arithmetic”) is respected and display a warning message if this is not the case.



8 Click Ok.

A “Save As ” window opens, enabling to save the current configuration.

9 Type `misrarules.txt` in the `C:\MISRA_results` directory.

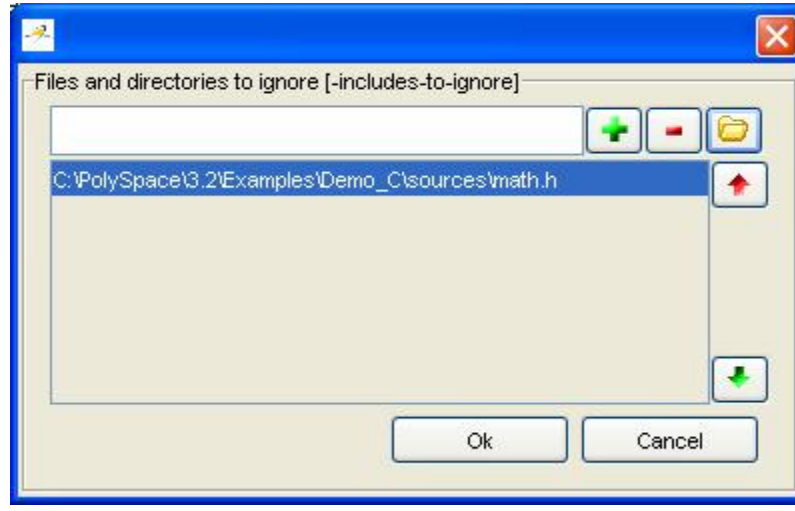
Discard Header Files from MISRA® Checking

You can disable MISRA verification on predefined files. For example, you may want to disable the MISRA C verification of `math.h`, included in `example.c`.

To discard header files:

1 Click … next to the -includes-to-ignore option.

The “Files and directories to ignore” window opens, enabling to disable the verification of MISRA C rules on selected files and directories.



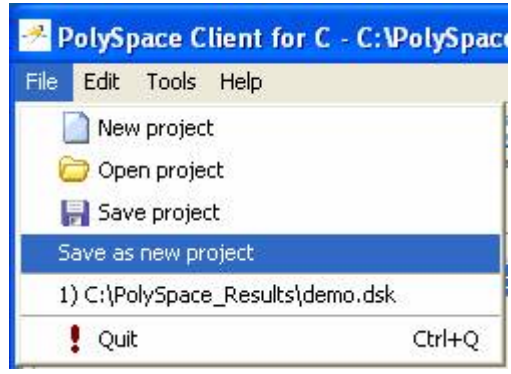
2 Select `math.h` using the browse button, then click Ok to close the window.

This file will not be checked.

This ends the setting up of the MISRA C checking phase.

Running the MISRA® Checker

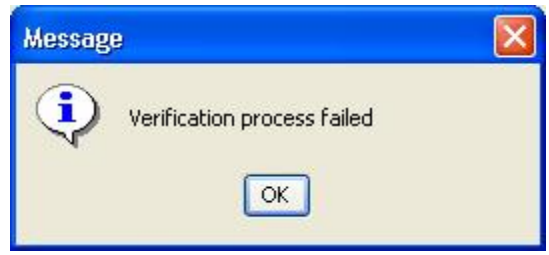
1 Before starting this analysis of `example.c` with MISRA C® checker, select "File>Save as new project" and choose another name for the current PolySpace™ project.



2 Click Execute to start the analysis.

During the new analysis performed on `example.c`, a new filter button is displayed - MISRA Log - in the log view. This button enables the user to filter out messages associated to the MISRA-C Checker.

At the end of the compilation process, PolySpace Desktop shows the following error message:



3 Click Ok .

The analysis has been interrupted, because the MISRA-C checker found that rule **16.3**, marked as “Error”, has not been respected. A list of MISRA-C errors and warning messages appears in the bottom window.

4 If we focus on the MISRA® log using appropriate filter MISRA Log , only messages associated to MISRA-C checker are displayed.

Also, the “Search in log” box permits to navigate into log file searching for an appropriate key word: a particular rule for instance.

```
Verifying C files ...
Verifying example.c
Verifying sources ...
Verifying example.c
 /sources/include.h:34 : MISRA-C ERROR : rule 16.3 (required) violated.
| Identifiers shall be given for all of the parameters in a function
prototype declaration.
example.c:97 : MISRA-C WARNING : rule 17.4 (required) violated.
| Array indexing shall be the only allowed form of pointer arithmetic.
example.c:113 : MISRA-C WARNING : rule 17.4 (required) violated.
| Array indexing shall be the only allowed form of pointer arithmetic.
example.c:117 : MISRA-C WARNING : rule 17.4 (required) violated.
| Array indexing shall be the only allowed form of pointer arithmetic.
example.c:121 : MISRA-C WARNING : rule 17.4 (required) violated.
| Array indexing shall be the only allowed form of pointer arithmetic.
```

5 Here, the recommendation is clear — an identifier is missing in a function prototype and must be added in “include.h” as required by MISRA-C rule **16.3**. Once this is done, you can re-launch the analysis.

Note You can also change the setting on rule **16.3** from “Error” to “Warning”, and launch the analysis again. The error message will change to “MISRA-C WARNING”, and the analysis will not stop after the ANSI® checking phase.

6 When no error remains after the ANSI checking phase, the analysis continues as described in step 1, and will give results as described in step 2.

Note A log file, located at the root of the `C:\PolySpace_Misra_Results` directory with `.log` as suffix, contains all messages displayed in the bottom window, including MISRA C messages. The format of the log file name is the following: `PolySpace_4_2_X_Y_`*`Name-Project_date_time.log`*.

5

Launching PolySpace™ Analysis Remotely

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`) 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

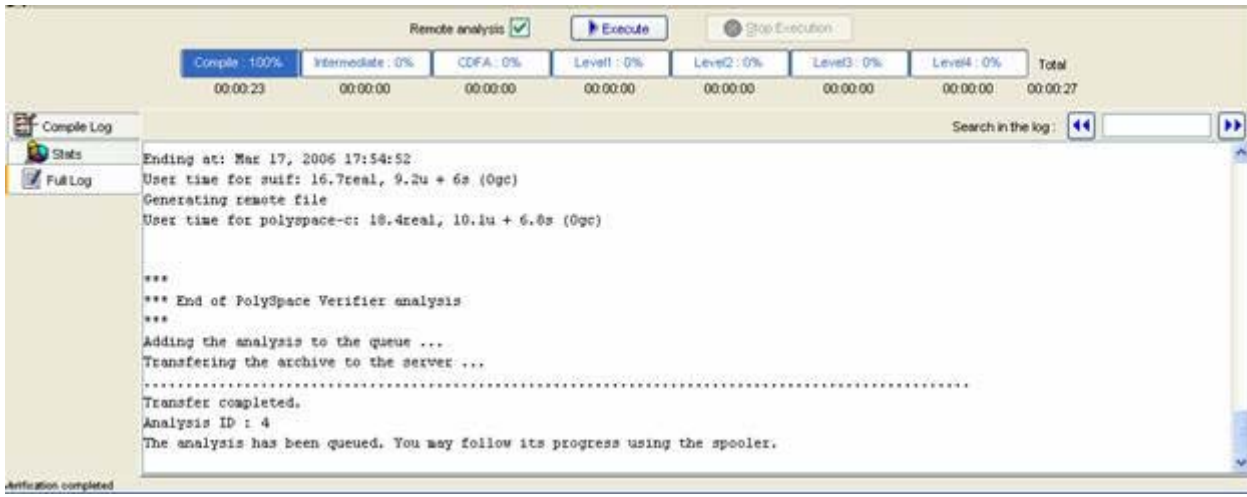
To launch an analysis remotely:

1. Set up an analysis as described in Chapter 2, “PolySpace™ Client — Analyzing a Single C File”, 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:



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.

Management of PolySpace™ Analysis in Remote: the PolySpace™ Spooler

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

To manage an analysis in the queue:

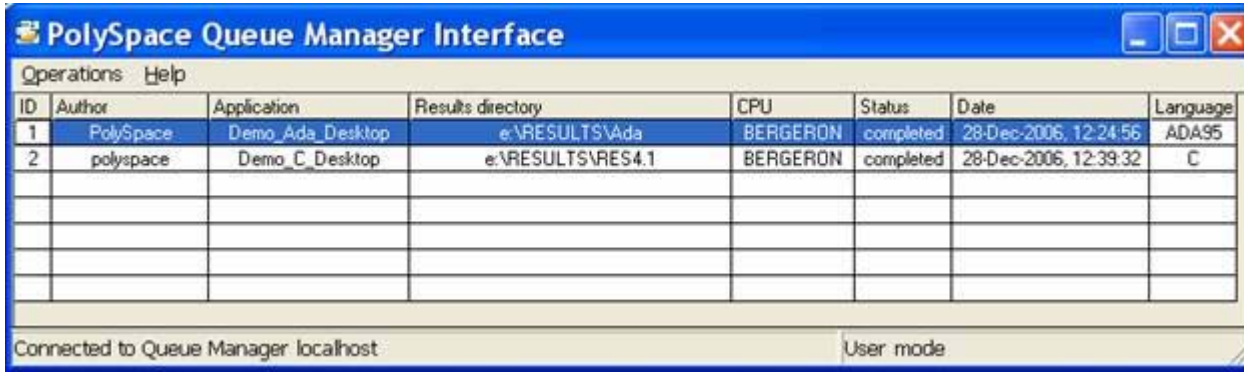
1 Open the PolySpace Spooler by either:

- Clicking on the short cut on your desktop PC

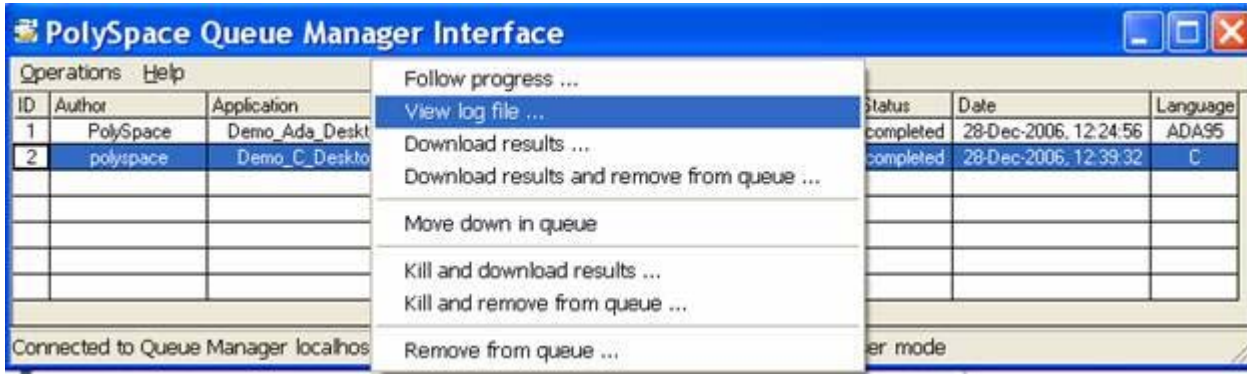
- Clicking on the icon

in the menu tab of the launcher.

The PolySpace Spooler appears.



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.

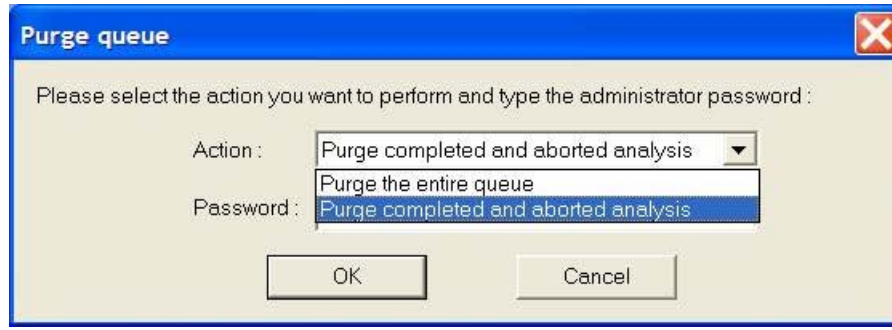
Note The results will be lost.

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

Note 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 of the queue manager or the default one.

Note By default the password is “administrator”.

Batch Commands

In this section...

Launching an Analysis in Batch

A set of commands allow the launching of analysis in batch.

All these commands begin with the following prefixes:

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

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

For example, `polyspace-remote-desktop-c -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-c.exe`

Managing an Analysis in Batch

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

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

On Windows platforms, 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 (see the 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

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®** — `/home/`*`username`*`/.PolySpace`
- **Windows®** — `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 an analysis using the PolySpace™ Client™ for C/C++ product, enabling or not the MISRA® Checker phase, 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 (in `PolySpaceInstallCommon\Docs\Manuals\`) or by clicking on  in PolySpace tools.