PolySpace
TECHNOLOGIES

PolySpace™ for C Getting Started R2007a+

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.

Table of Contents

Typographical conventions:

➪ The "➤" symbol indicates an action which must be performed by the user.
➪ `<PolySpaceInstallDir>` stands for the directory/folder name where the PolySpace products were installed.
➪ The "`Courier New`" font is used for mentioning data seen on the screen of the computer.

1. General Requirements

1.1. Computer Configuration

Please refer to PolySpace installation manual for the minimum hardware requirements.

The timing is the following:

- The installation of PolySpace products takes around 5 minutes (see the complete installation guide as available from the PolySpace installation CD-ROM in \Docs\Install\PolySpace_Install_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.

1.2. Structure of this document

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

Also, inside PolySpace Client and PolySpace Server, a PolySpace MISRA Checker is available, allowing the verification at compilation time of the rules recommended by the MISRA Consortium (more about MISRA Consortium at http://www.misra-c.com).

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

- In Step 1 we will analyze a simple file “example.c” by using PolySpace Client
- In Step 2.we will review the results obtained during Step 1 by using PolySpace Viewer
- In Step 3 we will use PolySpace MISRA Checker during the compilation phase of a PolySpace analysis.
- In Step 4, we will send an analysis remotely to a server.

2. Step 1:

PolySpace Client - Setting up and launching an analysis of a single C file

This paragraph 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 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 analysed source code: **ANSI compliance, portability** and **maintainability.**
2. Then, PolySpace seeks the main procedure. If none is found, PolySpace Client will generate one automatically. By default, this function will call all public functions of the file.
3. Finally, PolySpace proceeds with the code analysis phase, during which run time errors are detected and highlighted in the code.

2.1. Analysis prerequisites

Any analysis requires the following:

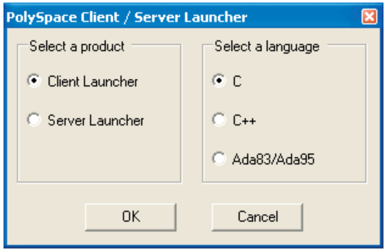
- PolySpace products and its related license file and dongle correctly installed;
- Source code files (in this case “`example.c`”) and all header files that it may directly or indirectly include. For this tutorial we will see later that we need two header files “`math.h`” and “`include.h`” in order to analyse “`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`”.

2.2. Setting up a PolySpace Client analysis

➤ Double-click on the PolySpace launcher icon:

PolySpace Launcher
Shortcut
2 KB

A window appears proposing to choose the product to be used for the analysis and the language of the file to be analyzed:


PolySpace Client / Server Launcher

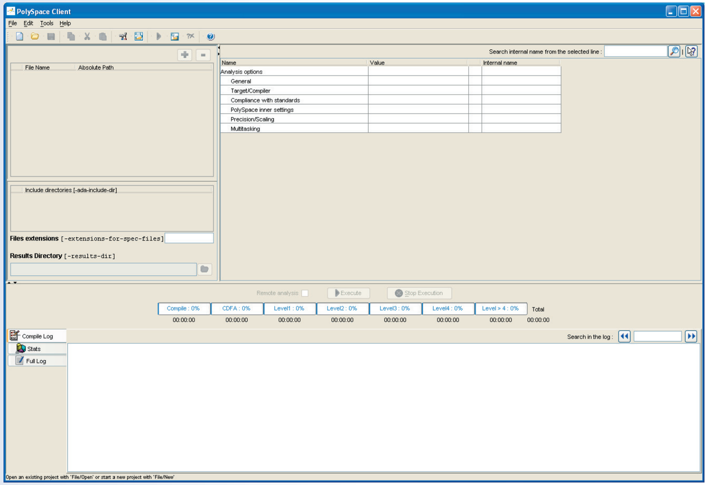
Select a product: [✓ Client Launcher] Server Launcher

Select a language: [✓ C] C++ Ada83/Ada95

If PolySpace is not installed for some languages, these choices of languages will be grayed out.

➤ Select "`Client Launcher`", language "C" and then, click on OK.

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



File Name	Absolute Path

Include directories [-ada-include-dir]

Files extensions `[-extensions-for-spec-files]`

Results Directory `[-results-dir]`

Search internal name from the selected line :

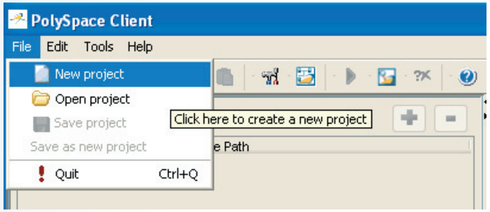
Name	Value		Internal name
Analysis options			
General			
Target/Compiler			
Compliance with standards			
PolySpace inner settings			
Precision/Scaling			
Multitasking			

Compile : 0%	CDFA : 0%	Level1 : 0%	Level2 : 0%	Level3 : 0%	Level4 : 0%	Level > 4 : 0%	Total
00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00

Open an existing project with 'File/Open' or start a new project with 'File/New'

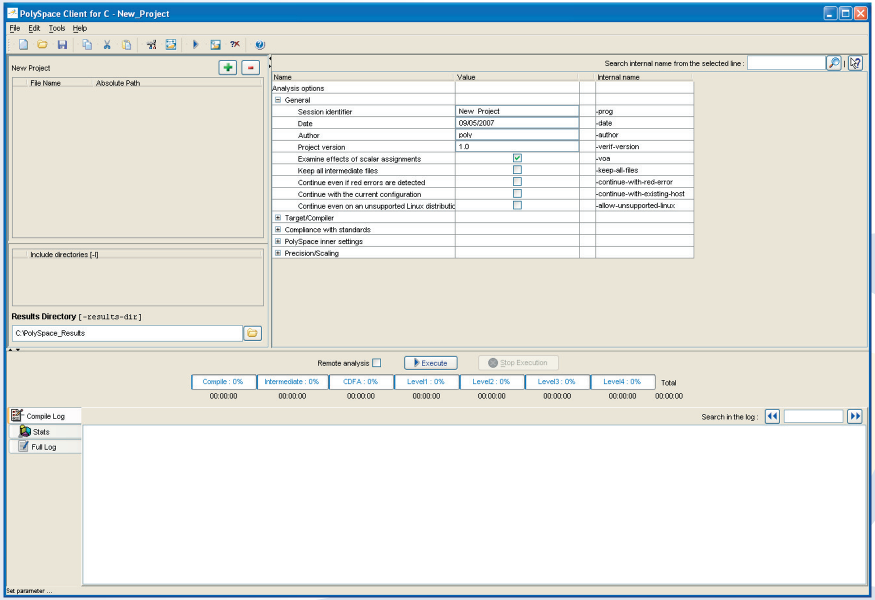
Step 1: PolySpace Client - Setting up and launching an analysis of a single C file

➤ Click on `File/New Project` to start an analysis:



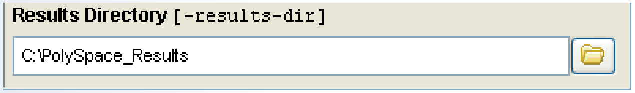
The PolySpace Client New Project window opens. It contains four sections:

- 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;
- The bottom area allows following the execution and progress of the analysis.



Name	Value	Internal name
Analysis options		
⊟ General		
Session identifier	New Project	-prog
Date	09/05/2007	-date
Author	poly	-author
Project version	1.0	-verif-version
Examine effects of scalar assignments	✓	-voa
Keep all intermediate files	☐	-keep-all-files
Continue even if red errors are detected	☐	-continue-with-red-error
Continue with the current configuration	☐	-continue-with-existing-host
Continue even on an unsupported Linux distributic	☐	-allow-unsupported-linux
⊞ Target/Compiler		
⊞ Compliance with standards		
⊞ PolySpace inner settings		
⊞ Precision/Scaling		

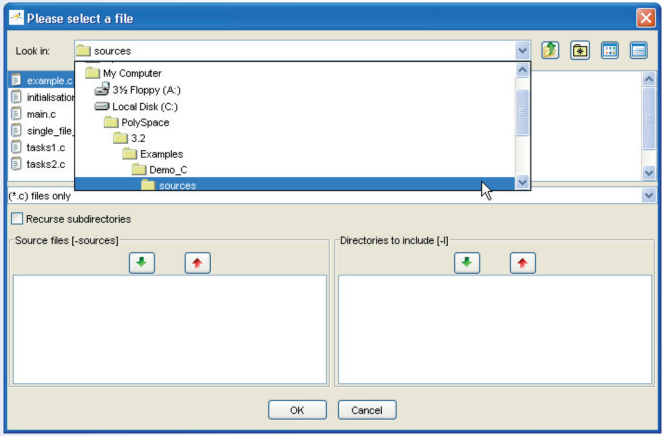
➤ Start by updating the result directory name by clicking on the browse button :


Results Directory [-results-dir]

C:\PolySpace_Results

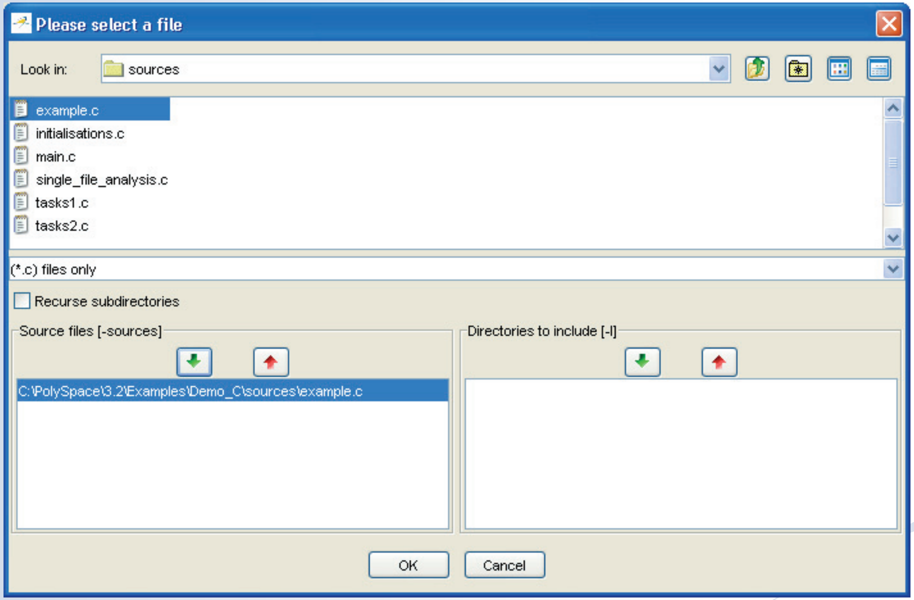
This directory is the one where PolySpace Client will store the results of the analysis. In this Getting Started, we will choose the default directory: "C:\PolySpace_Results".

➤ 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 analyse.



➤ In the "Look in" section, click on and select "<PolySpaceInstallDir>\Examples\Demo_C\sources". A list of files appears in the box. The default <PolySpaceInstallDir> is C:\PolySpace\PolySpaceForCandCPP.

➤ Select "example.c" and click on in the "Source files [-sources]" section (bottom right) of the window. The file is now listed among the source files to be analyzed.



C:\PolySpace\3.2\Examples\Demo_C\sources\example.c

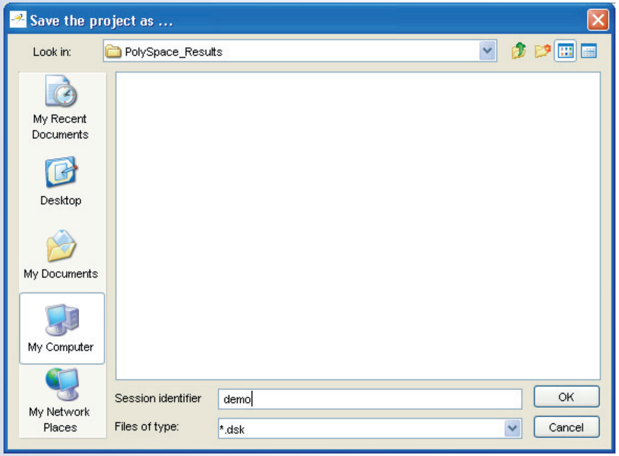
➤ Click on OK to go back to the “`PolySpace Client for C - New_Project`” window.

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

2.3. PolySpace Client: running the analysis

➤ Click on Execute 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 "`demo`" and save it in the PolySpace result directory. The full name of that file will be "`demo.dsk`".

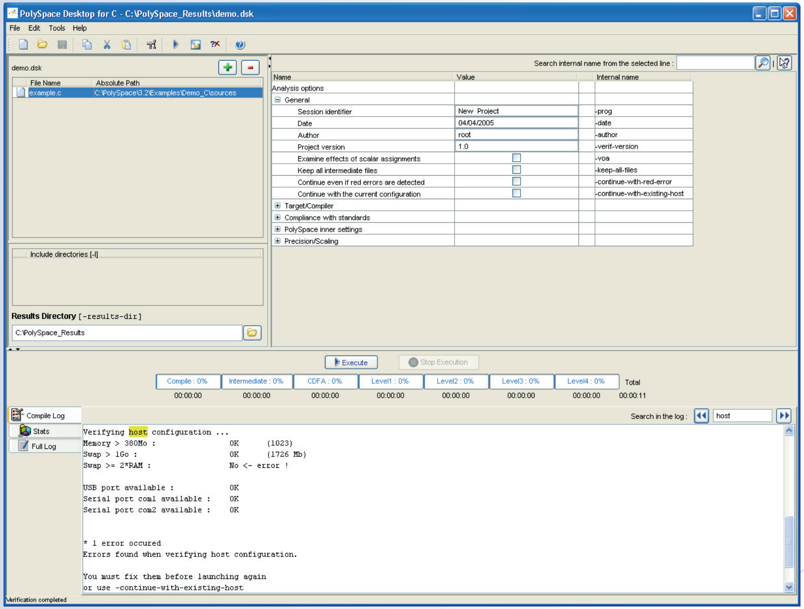


Look in: PolySpace_Results

Session identifier: demo

Files of type: *.dsk

➤ Click on OK to go back to the "`PolySpace Client for C – New_Project`" window and click again on Execute to proceed forward.



File Name	Absolute Path
example.c	C:\PolySpace\3.2\Examples\Demo_C\sources

Results Directory [-results-dir]

C:\PolySpace_Results

Search internal name from the selected line :

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	☐	-voa
Keep all intermediate files	☐	-keep-all-files
Continue even if red errors are detected	☐	-continue-with-red-error
Continue with the current configuration	☐	-continue-with-existing-host
⊞ Target/Compiler		
⊞ Compliance with standards		
⊞ PolySpace inner settings		
⊞ Precision/Scaling		

Compile : 0%	Intermediate : 0%	CDFA : 0%	Level1 : 0%	Level2 : 0%	Level3 : 0%	Level4 : 0%	Total
00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:11

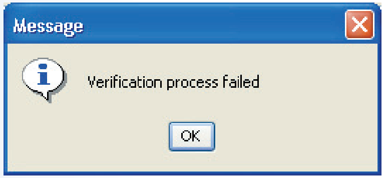
```
Verifying host configuration ...
Memory > 380Mo :               OK      (1023)
Swap > 1Go :                   OK      (1726 Mb)
Swap >= 2*RAM :                No <- error !

USB port available :           OK
Serial port com1 available :   OK
Serial port com2 available :   OK


* 1 error occured
Errors found when verifying host configuration.

You must fix them before launching again
or use -continue-with-existing-host
```

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 use the Stop Execution button - Stop Execution - to interrupt the analysis but it is not part of the current tutorial.

2.3.1. Parsing errors during preliminary PolySpace analysis stages

After some checks, PolySpace will show an error message:

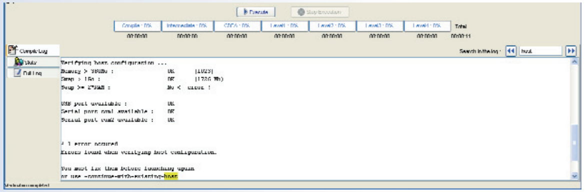


```
Message
Verification process failed
OK
```

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 meets the minimal hardware requirements described in section "`General Requirements`". A message similar to the one below would be displayed in the bottom part of the graphical interface:



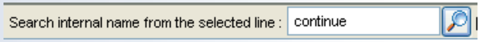
```
Verifying host configuration ...
Memory > 980Mo :                OK      [1023]
Swap > 1Go :                    OK      [1726 Mo]
Swap >= 2*RAM :                 No <  error !

USB port available :            OK
Serial port com1 available :    OK
Serial port com2 available :    OK


# 1 error occured
Errors found when verifying host configuration.

You must fix them before launching again
or use -continue-with-existing-host
```

➤ To help you understand the issue, you can search into the log file. Type "`host`" in the "`Search in the log:`" box and click on ◂◂ to check whether the error corresponds to a hardware recommendation problem.

If you have a problem related to host configuration, in order to continue analysis, you can either:

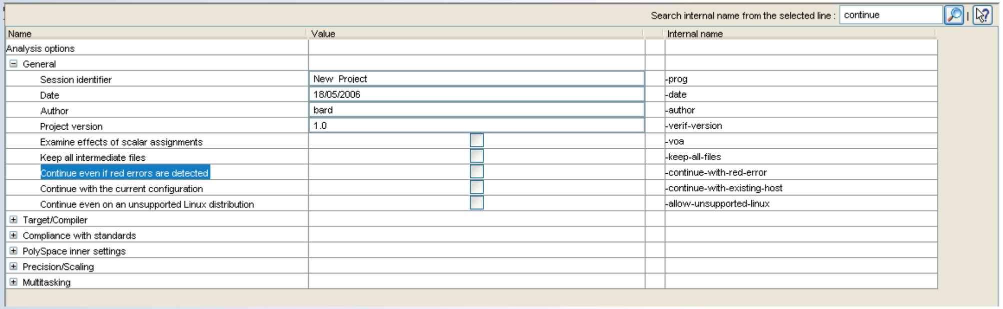
- upgrade your computer to meet the minimal requirements
- or use the `-continue-with-existing-host` option which overrides the initial check for minimal hardware configuration.

➤ To set up the `-continue-with-existing-host` option, please type "`continue`" in the "`Search internal name from the selected line`" box at the top right of the window



Search internal name from the selected line : continue

➤ Then click on . It will show all options containing the word "`continue`" as shown below:

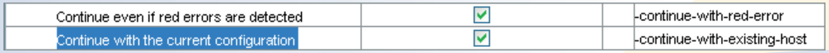


Search internal name from the selected line : continue

Name	Value	Internal name
Analysis options		
⊟ General		
Session identifier	New Project	-prog
Date	18/05/2006	-date
Author	bard	-author
Project version	1.0	-verif-version
Examine effects of scalar assignments	☐	-voa
Keep all intermediate files	☐	-keep-all-files
Continue even if red errors are detected	☐	-continue-with-red-error
Continue with the current configuration	☐	-continue-with-existing-host
Continue even on an unsupported Linux distribution	☐	-allow-unsupported-linux
⊞ Target/Compiler		
⊞ Compliance with standards		
⊞ PolySpace inner settings		
⊞ Precision/Scaling		
⊞ Multitasking		

➤ Check the box ✓ in the "Value" column that is associated to the "`-continue-with-existing-host`" line as shown below.

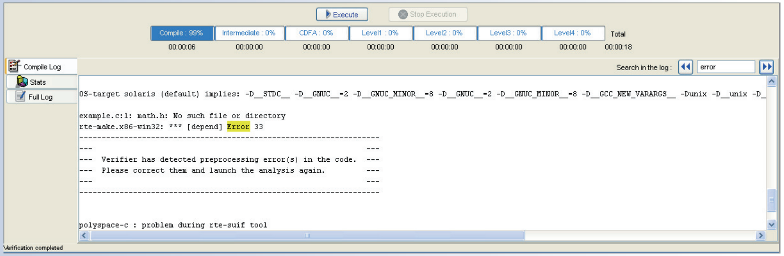
➤ **It is also recommended** to select the `–continue-with-red-error` option. Indeed, "`example.c`" contains – on purpose - code with some definite errors, later called red errors. This option allows you to continue the analysis even if red errors are detected. Otherwise, the analysis would just stop after the detection of the first of these errors.



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

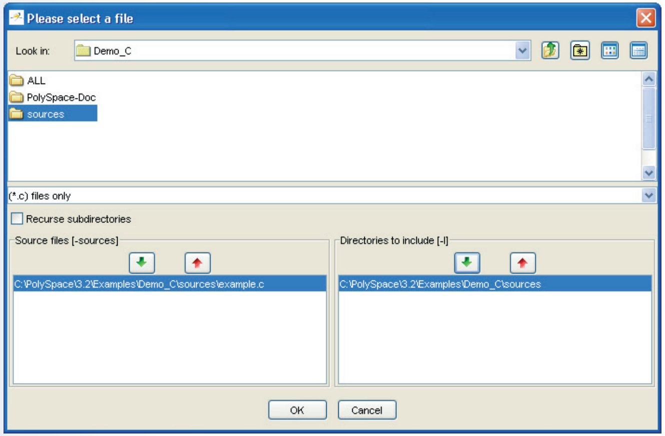
Another cause of error may be that PolySpace Client misses some information about header files.



```
OS-target solaris (default) implies: -D__STDC__ -D__GNUC__=2 -D__GNUC_MINOR__=8 -D__GNUC__=2 -D__GNUC_MINOR__=8 -D__GCC_NEW_VARARGS__ -Dunix -D__unix -D_

example.c:1: math.h: No such file or directory
rte-make.x86-win32: *** [depend] Error 33
-----------------------------------------------------------------------
---                                                                 ---
---  Verifier has detected preprocessing error(s) in the code.      ---
---  Please correct them and launch the analysis again.             ---
---                                                                 ---
-----------------------------------------------------------------------



polyspace-c : problem during rte-suif tool
```

In the tutorial, as shown on the screen capture opposite, the file named "`math.h`" can not 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 "`example.c`" file analysis, the related "`math.h`" file is located in the same directory as the C file: `<PolySpaceInstall>\Examples\Demo_C`.



C:\PolySpace\3.2\Examples\Demo_C\sources\example.c

C:\PolySpace\3.2\Examples\Demo_C\sources

➤ Open the "`Please select a file`" window by using the button (right of the "`demo.dsk`" label in the top right of the interface)

➤ Select "`<PolySpaceInstallDir>\Examples\Demo_C\sources`", where "`math.h`" is located.

➤ Click on in the "`Directories to include [-I]`" section and then close the window using OK

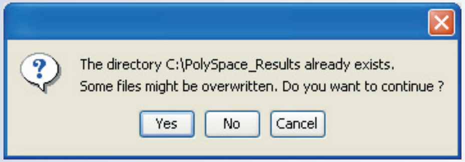
Notes:

1. The other header file needed "`include.h`" is located in same directory.
2. It is also possible to drag a directory and drop it in the "`include directories [-I]`" part (top left of PolySpace Client) without using the "`Please select a file`" window.

2.3.2. Progression of the analysis

➤ Click on Execute to restart the analysis.

Some results may have already been written in the “`C:\PolySpace_Results`” directory, because of a previous click on Execute. Therefore a window opens to check whether you want to overwrite them:



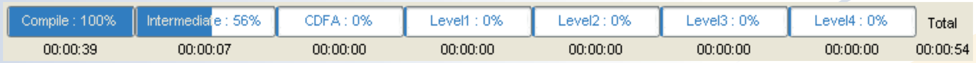
The directory C:\PolySpace_Results already exists.
Some files might be overwritten. Do you want to continue ?

Yes No Cancel

In our example, this is what we want to do. Click on Yes

Note: closing the PolySpace Client window will not stop the PolySpace analysis. If you wish to stop it, click on Stop Execution (you will be asked for confirmation). If the window is closed without stopping the analysis, the analysis continues in the background. Opening again PolySpace Client with the same project automatically updates the analysis with its current status.

The progress bar allows following the progress of the analysis:

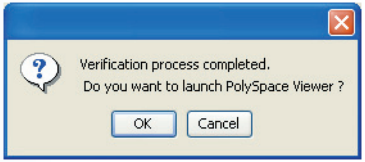


Compile : 100%	Intermediate : 56%	CDFA : 0%	Level1 : 0%	Level2 : 0%	Level3 : 0%	Level4 : 0%	Total
00:00:39	00:00:07	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:54

A progress report may be obtained by clicking on Compile Log for the compilation phase, or Stats for the full analysis in the bottom part of the window. Click on Full Log to get additional information about the current analysis (list of options, stubbed functions, functions used during main construction, checks found after each phase, etc.). Click on the icon to refresh the summary.

2.3.3. End of the analysis

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



Verification process completed.
Do you want to launch PolySpace Viewer ?

➤ Click on OK, and go to the next section of the tutorial to view the results.
If you click on Cancel, and if no other analyses is running, you can access the results via the icon in the title bar.

3. Step 2: PolySpace Viewer - Exploration of results

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 "`example.c`" performed during Step 1.

You can access the results of any successful analysis by double clicking on the PolySpace Viewer icon:

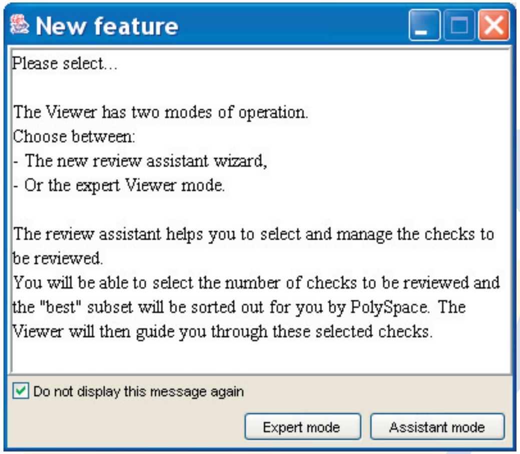
If the OK button has been clicked at the end of the previous analysis (see previous section), PolySpace Viewer automatically opens results.

3.1. PolySpace Viewer modes of operation

The first time the PolySpace Viewer is opened, a window will appear to describe the different modes of operation.

- In "`Expert mode`", all checks can be seen.
  The number and categories of checks to be reviewed as well as the order in which to review them can be chosen by the user (See next section).
- In "`Assistant mode`", the rules of the review follows a methodology selected by PolySpace. The "best" subset of checks will be automatically selected and sorted out. The PolySpace Viewer will then guide the user through these selected checks.

➤ For this tutorial, please untick
"`Do not display this message again`"
and then click on "`Expert mode`".


New feature

Please select...

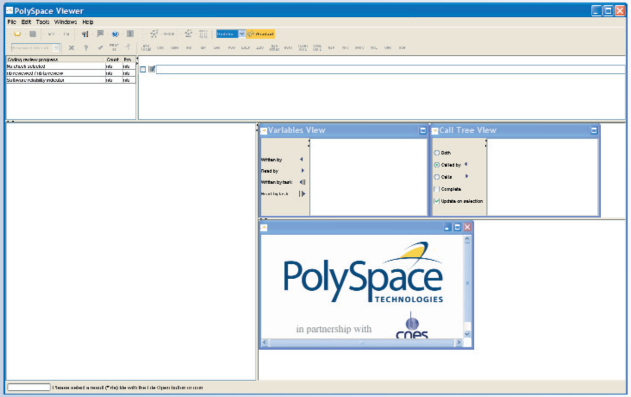
The Viewer has two modes of operation.
Choose between:
- The new review assistant wizard,
- Or the expert Viewer mode.

The review assistant helps you to select and manage the checks to be reviewed.
You will be able to select the number of checks to be reviewed and the "best" subset will be sorted out for you by PolySpace. The Viewer will then guide you through these selected checks.

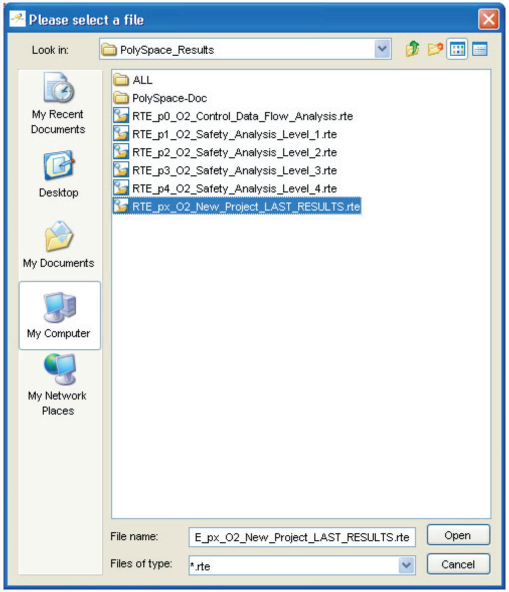
☑ Do not display this message again

Note: The mode of operation may be changed later in PolySpace Viewer.

3.2. Download results



After having clicked on "`Expert mode`" the PolySpace Viewer window looks like the one on the left:



Look in: PolySpace_Results

ALL
PolySpace-Doc
RTE_p0_O2_Control_Data_Flow_Analysis.rte
RTE_p1_O2_Safety_Analysis_Level_1.rte
RTE_p2_O2_Safety_Analysis_Level_2.rte
RTE_p3_O2_Safety_Analysis_Level_3.rte
RTE_p4_O2_Safety_Analysis_Level_4.rte
RTE_px_O2_New_Project_LAST_RESULTS.rte

File name: E_px_O2_New_Project_LAST_RESULTS.rte
Files of type: *.rte

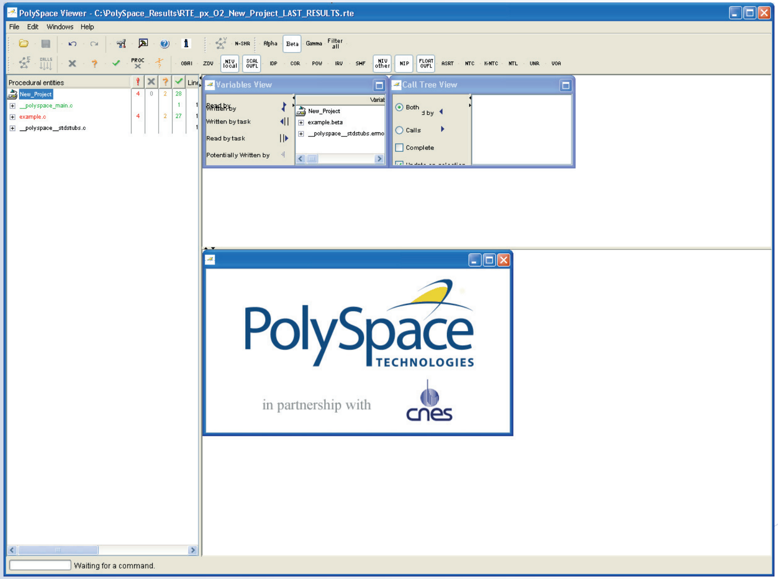
➤ Click "`File>Open`" to load your result files.
If you did not perform the analysis, you can still review the results by opening the following file:
`<PolySpaceInstallDir>\Examples`
`\Demo_C\RTE_px_O2_Demo_C_LAST_RESULTS.rte`

➤ Then click on Open to proceed with further steps

Note: The `RTE_px_O2_Demo_C_LAST_RESULTS.rte` is a "link" to the best results for the analysis in term of precision: `RTE_p4_O2_Safety_Analysis_Level4.rte`.
Other results have lower precision.

3.3. Reviewing PolySpace results in "Expert" mode

After loading the results in "`Expert`" mode, PolySpace Viewer window looks like below:



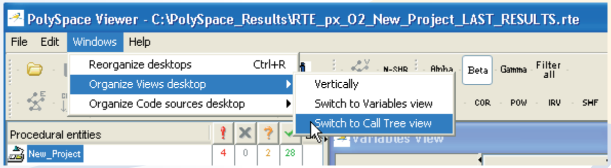
1. On the left is the run-time error view (RTE or "`Procedural Entities`" View). It displays the list of files analyzed in the "`Procedural entities`" column.

2. In the bottom right area is the source code view. Each operation checked is displayed using meaningful colour 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.

3. The two windows just below the tool bar display details about the currently reviewed check (when the check has been selected):

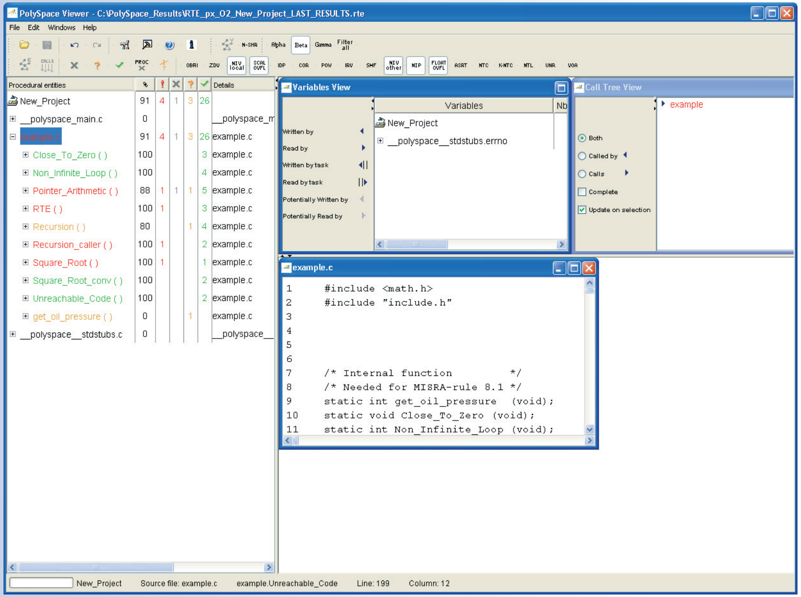
4. The top right area is used for displaying both control and data flow results.
You can switch from one view to the other by using the "Windows" menu:


PolySpace Viewer - C:\PolySpace_Results\RTE_px_02_New_Project_LAST_RESULTS.rte
File Edit Windows Help
Reorganize desktops Ctrl+R
Organize Views desktop
Organize Code sources desktop
Vertically
Switch to Variables view
Switch to Call Tree view
Procedural entities
New_Project

3.3.1. Procedural Entities view

Each file and underlying functions in the RTE view is colorized according to the most critical error found:

- __polyspace_main.c. This file contains the main which was automatically generated by PolySpace. 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 and is thus not colored. It contains stubs of standard functions part of `libc` library used in example.c.


PolySpace Viewer - C:\PolySpace_Results\RTE_px_O2_New_Project_LAST_RESULTS.rte

Procedural entities	%	!	✗	?	✓	Details
New_Project	91	4	1	3	26	
__polyspace_main.c	0					__polyspace_m
example.c	91	4	1	3	26	example.c
Close_To_Zero ( )	100				3	example.c
Non_Infinite_Loop ( )	100				4	example.c
Pointer_Arithmetic ( )	88	1	1	1	5	example.c
RTE ( )	100	1			3	example.c
Recursion ( )	80			1	4	example.c
Recursion_caller ( )	100	1			2	example.c
Square_Root ( )	100	1			1	example.c
Square_Root_conv ( )	100				2	example.c
Unreachable_Code ( )	100				2	example.c
get_oil_pressure ( )	0			1		example.c
__polyspace__stdstubs.c	0					__polyspace__

```
#include <math.h>
#include "include.h"




/* Internal function          */
/* Needed for MISRA-rule 8.1 */
static int get_oil_pressure  (void);
static void Close_To_Zero (void);
static int Non_Infinite_Loop (void);
```

New_Project   Source file: example.c   example.Unreachable_Code   Line: 199   Column: 12

➤ Select the “Beta” filter in the combo menu in the toolbar and click once on + sign at the left of “`example.c`” to find out more about this file. “`example.c`” is expanded and the list of functions defined into it is displayed. The functions in red or grey (`Pointer_Arithmetic()`, `Square_Root()`, etc.) have code sections that need to be inspected first because they are definite diagnosis of PolySpace (either runtime errors or dead code).

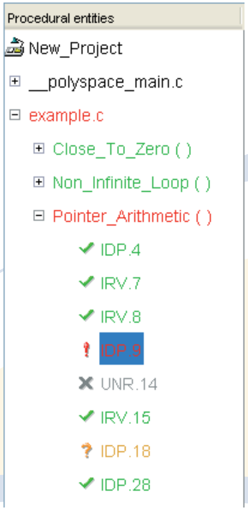
The columns (%, !, ✓, ✗, ?,...) provide information about run-time errors found in each function:

- The % column indicates the Software reliability indicator in percentage (level of proof). 100% means a complete reliability of the code for the category checked by PolySpace with the hypothesis taken for the analysis.
- The ! column indicates the number of definite run-time errors or reds,
- The ? column indicates the number of unproven or oranges (may be run-time errors that do not occur systematically),
- The ✓ column indicates the number of safe operations or greens
- The ✗ column indicates the number of unreachable instructions or grey code sections.
- The Reviewed column allows marking reviewed checks.

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

First example of runtime error found by PolySpace: Memory Corruption

➤ 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 checked within the "`Pointer_Arithmetic()`" function.

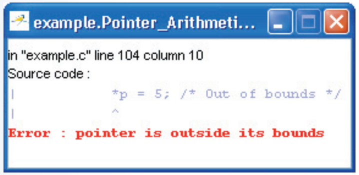


Procedural entities

New_Project
⊞ __polyspace_main.c
⊟ example.c
  ⊞ Close_To_Zero ( )
  ⊞ Non_Infinite_Loop ( )
  ⊟ Pointer_Arithmetic ( )
    ✓ IDP.4
    ✓ IRV.7
    ✓ IRV.8
    ! IDP.9
    ✗ UNR.14
    ✓ IRV.15
    ? IDP.18
    ✓ IDP.28

➤ Click on the red "IDP.9" item - which stands for Illegal **D**e-referenced **P**ointer -, to precisely locate this error in the source code. The bottom right section is updated showing the location of the "IDP.9" item.

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

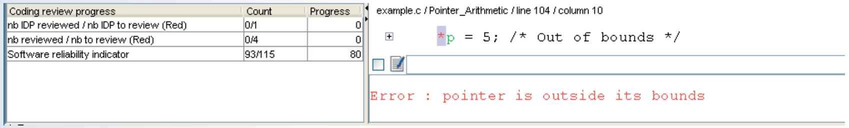


in "example.c" line 104 column 10
Source code :

```
|            *p = 5; /* Out of bounds */
|            ^
Error : pointer is outside its bounds
```

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. The left one gives some statistics about all the IDP in the analysis:

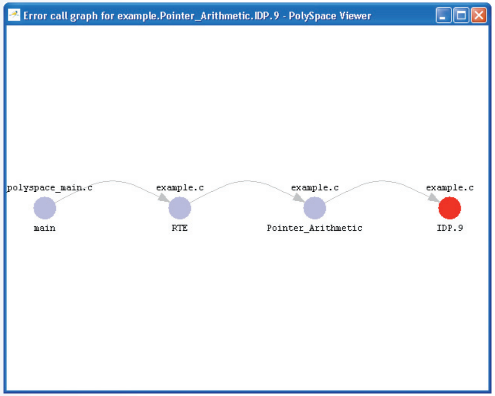


Coding review progress	Count	Progress
nb IDP reviewed / nb IDP to review (Red)	0/1	0
nb reviewed / nb to review (Red)	0/4	0
Software reliability indicator	93/115	80

example.c / Pointer_Arithmetic / line 104 / column 10

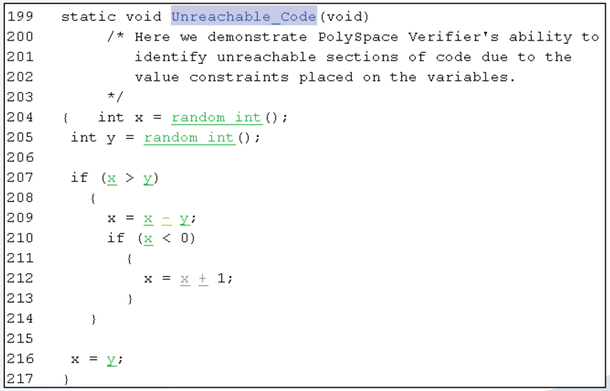
```
*p = 5; /* Out of bounds */

Error : pointer is outside its bounds
```

➤ You can also see the calling sequence leading to that particular red code section. To do so, select the "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 found by PolySpace: Unreachable code

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



```
static void Unreachable_Code(void)
      /* Here we demonstrate PolySpace Verifier's ability to
         identify unreachable sections of code due to the
         value constraints placed on the variables.
      */
{   int x = random_int();
 int y = random_int();

 if (x > y)
   {
      x = x - y;
      if (x < 0)
        {
          x = x + 1;
        }
   }

 x = y;
}
```

3.3.2. Colours in the source code view

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

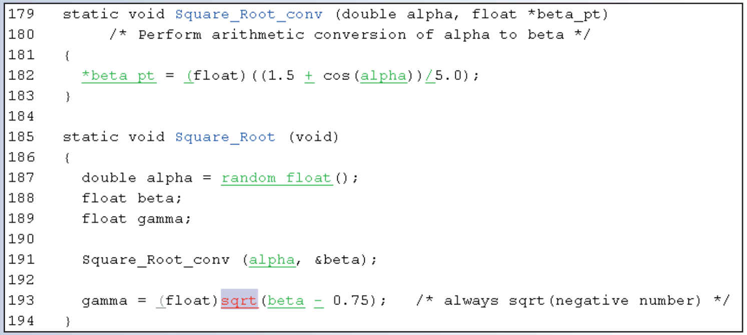
1. Red: A link to the error message associated to the error which occurs at every execution.
2. Orange: A link to an unproven message – an error may occur sometimes.
3. Grey: A link to a check shown as unreachable code. The error message is in grey.
4. Green: A link to a VOA (Value on Assignment) or an error condition that will never occur in the list of verifications made by PolySpace.
5. 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;`.
6. Blue: text highlighting the keyword "`procedure`" and "`function`".
7. Underlined blue: A link to a global variable in the "Global variable View".

3.3.3. 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.

In a first example of the second set: Arithmetic error

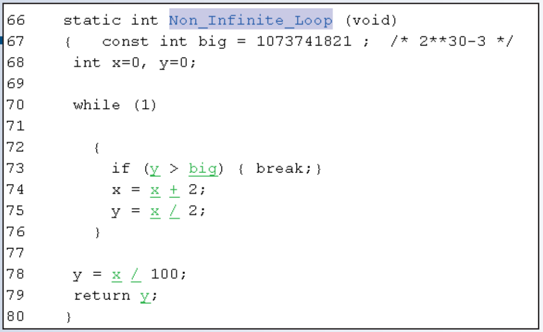
➤ Click on ⊞ to expand "`Square_Root()`" function. You can see the source code view in the bottom right. You can also display the call tree for that function by using the "`Windows`" menu (see previous paragraph). "`Square_Root()`" is called by RTE function from "`example.c`". It is displayed as "`example.RTE`" 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).



```
static void Square_Root_conv (double alpha, float *beta_pt)
     /* Perform arithmetic conversion of alpha to beta */
{
  *beta_pt = (float)((1.5 + cos(alpha))/5.0);
}

static void Square_Root (void)
{
  double alpha = random_float();
  float beta;
  float gamma;

  Square_Root_conv (alpha, &beta);

  gamma = (float)sqrt(beta - 0.75);   /* always sqrt(negative number) */
}
```

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 the `–voa` option when launching the analysis, PolySpace can give more information about the range on scalar assignments



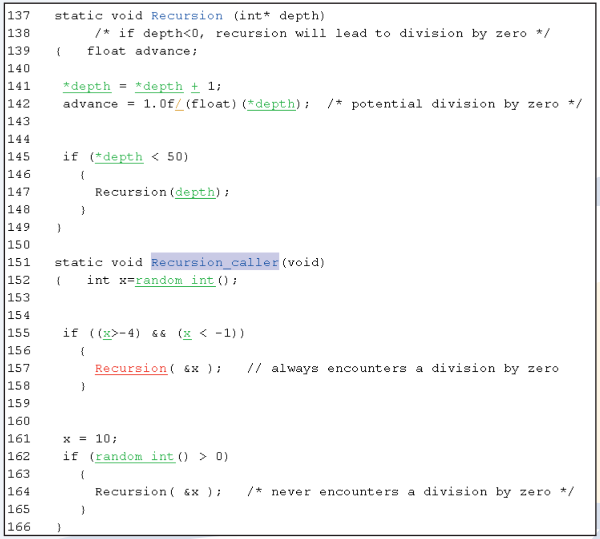
```
static int Non_Infinite_Loop (void)
{   const int big = 1073741821 ;  /* 2**30-3 */
 int x=0, y=0;

 while (1)

   {
     if (y > big) { break;}
     x = x + 2;
     y = x / 2;
   }

 y = x / 100;
 return y;
}
```

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 local 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.

Third example of the second set:
Errors due to wrong value passed to a function call

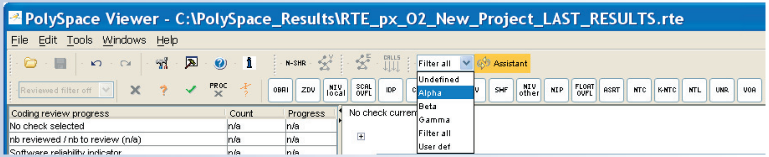
➤Select “Recursion_caller()”: The first call to Recursion is in red because when a negative parameter is passed, Recursion encounters a division by zero (See the “Recursion” function). PolySpace also checks recursive constructs:



```
static void Recursion (int* depth)
     /* if depth<0, recursion will lead to division by zero */
{   float advance;

 *depth = *depth + 1;
 advance = 1.0f/(float)(*depth);  /* potential division by zero */


 if (*depth < 50)
   {
     Recursion(depth);
   }
}

static void Recursion_caller(void)
{   int x=random_int();


 if ((x>-4) && (x < -1))
   {
     Recursion( &x );   // always encounters a division by zero
   }


 x = 10;
 if (random_int() > 0)
   {
     Recursion( &x );   /* never encounters a division by zero */
   }
}
```

3.3.4. 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 "`Alpha`", "`Beta`", "`Gamma`", "`User def`" and "`Filter All`" that you can choose depending on your development process:


PolySpace Viewer - C:\PolySpace_Results\RTE_px_02_New_Project_LAST_RESULTS.rte
File Edit Tools Windows Help
N-SHR
CALLS
Filter all
Assistant
Undefined
Alpha
Beta
Gamma
Filter all
User def
Reviewed filter off
PROC
OBAI ZDV NIV local SCAL OVFL IDP
SHF NIV other NIP FLOAT OVFL ASRT NTC K-NTC NTL UNR VOA
No check curren

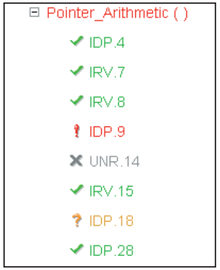
Coding review progress	Count	Progress
No check selected	n/a	n/a
nb reviewed / nb to review (n/a)	n/a	n/a
Software reliability indicator	n/a	n/a

➤ Select "`Gamma`" to get all the "red" and "grey" code `sections`. It is mainly used during the earliest development stages to focus quickly on critical bugs.

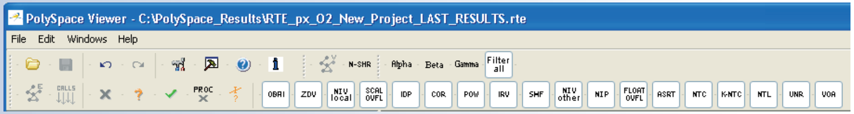
➤ To illustrate the use of these filters, we will focus on the Pointer Arithmetic function that we have examined in the 3.3.2 section. Select "`-`" to hide the checks related to "`Pointer_Arithmetic()`".

This list of acronyms - for type of operations checked - shows what kind of errors PolySpace automatically looked for. The "`Beta`" level highlights checks that could cause a processor halt, memory corruptions or overflows.

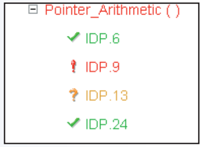
➤ Select "`Beta`" mode (which is the default mode). Select again "`Pointer_Arithmetic()`" in the "Procedural entities" view and then, click on the + sign to expand it and get the list of checks:



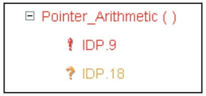
To get the comprehensive list of operations checked by PolySpace, you can switch to Alpha mode. You may also want to use filters to focus on particular categories of errors. Those filters are located at the top of the PolySpace Viewer window:


PolySpace Viewer - C:\PolySpace_Results\RTE_px_02_New_Project_LAST_RESULTS.rte
File Edit Windows Help
N-SHR
Alpha Beta Gamma Filter all
CALLS
PROC
OBAI ZDV NIV local SCAL OVFL IDP COR POW IRV SHF NIV other NIP FLOAT OVFL ASRT NTC K-NTC NTL UNR VOA

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


⊟ Pointer_Arithmetic ( )
✓ IDP.6
! IDP.9
? IDP.13
✓ IDP.24

➤ Click on Filter all (top of the window) to suppress all checks and click on IDP. You will get list of checks containing only IDP (**I**llegal **D**ereference **P**ointers) reds, unproven or greens:


⊟ Pointer_Arithmetic ( )
! IDP.9
? IDP.18

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

3.3.5. 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`").

3.4. Methodological assistant

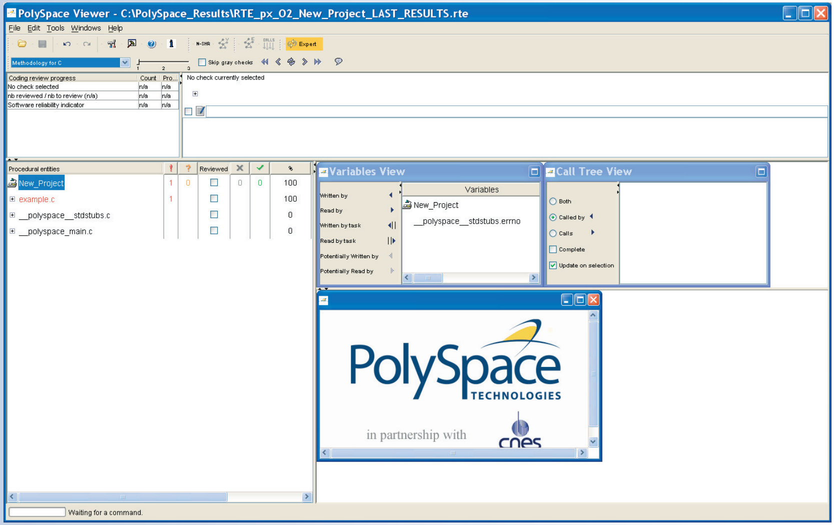
After this first usage of PolySpace Viewer, some simple questions remain:

- Do all checks need to be reviewed?
- If not, what are the checks to review?
- How many?
- What is the best order?

The Methodological assistant answers to all theses questions. It helps to select and manage the checks to be reviewed. It selects a "best" subset and sorts checks out.
The Assistant mode in the PolySpace Viewer will then guide you through these selected checks.

➤ If the PolySpace Viewer is still open, close it by clicking on the upper right ☒ symbol, open it again, load the same results and chose the "`Assistant`" mode.

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



PolySpace Viewer - C:\PolySpace_Results\RTE_px_O2_New_Project_LAST_RESULTS.rte

File Edit Tools Windows Help

Expert

Methodology for C — Skip gray checks

Coding review progress	Count	Pro...
No check selected	n/a	n/a
nb reviewed / nb to review (n/a)	n/a	n/a
Software reliability indicator	n/a	n/a

No check currently selected

Procedural entities	!	?	Reviewed	✗	✓	%
New_Project	1	0	☐	0	0	100
example.c	1		☐			100
__polyspace__stdstubs.c			☐			0
__polyspace_main.c			☐			0

Variables View: Written by, Read by, Written by task, Read by task, Potentially Written by, Potentially Read by

Variables
New_Project
__polyspace__stdstubs.errno

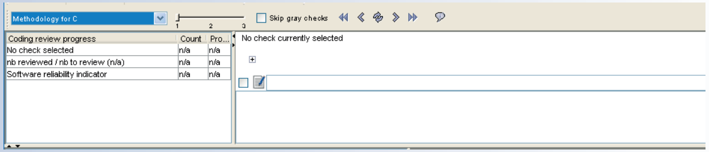
Call Tree View: ○ Both, [● Called by], ○ Calls, ☐ Complete, [✓ Update on selection]

PolySpace TECHNOLOGIES in partnership with cnes

Waiting for a command.

3.4.1. Assistant dashboard

The second line of buttons on the toolbar and the two views just below are used to navigate between the checks selected by PolySpace:



Methodology for C — Skip gray checks

Coding review progress	Count	Pro...
No check selected	n/a	n/a
nb reviewed / nb to review (n/a)	n/a	n/a
Software reliability indicator	n/a	n/a

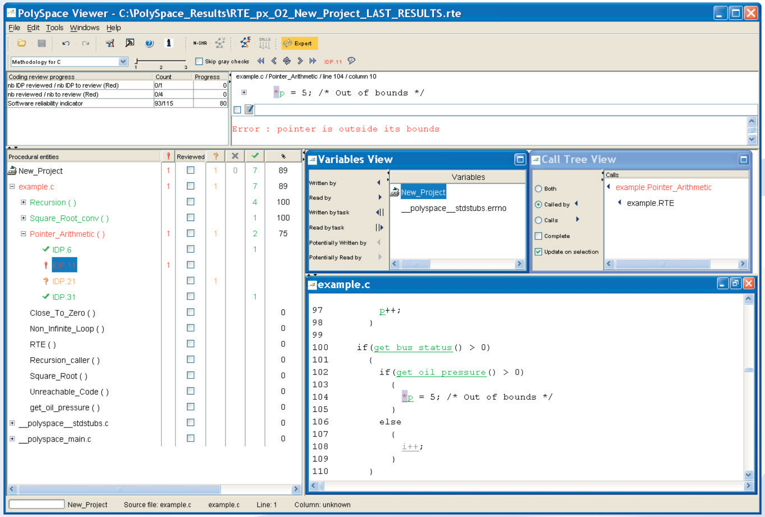
No check currently selected

PolySpace Viewer has also been updated as follows:

1. Now, in the "Procedural Entities" view the list of files analyzed is sorted out according to the methodological assistant used.
2. In the source code view, each operation will be sorted out according to the PolySpace methodology in the following order:
   - Red: The methodological assistant browses all red errors.
   - Gray: The methodological assistant browses unreachable code depending on the radio button "`Skip gray checks`".
   - Orange: The methodological assistant chooses and reviews the "best" unproven operations - those that are the most probably actual errors.

➤ Click on to go to the next check.

PolySpace Viewer has been refreshed with the first check selected by the methodological assistant:


PolySpace Viewer - C:\PolySpace_Results\RTE_px_O2_New_Project_LAST_RESULTS.rte
File Edit Tools Windows Help
N-SHR
CALLS
Expert
Methodology for C
1 2 3
Skip gray checks
IDP.11
Variables View
Call Tree View
example.c

Coding review progress	Count	Progress
nb IDP reviewed / nb IDP to review (Red)	0/1	0
nb reviewed / nb to review (Red)	0/4	0
Software reliability indicator	93/115	80

example.c / Pointer_Arithmetic / line 104 / column 10

```
*p = 5; /* Out of bounds */
```

```
Error : pointer is outside its bounds
```

Procedural entities	!	Reviewed	?	✗	✓	%
New_Project	1	☐	1	0	7	89
example.c	1	☐	1		7	89
Recursion ( )		☐			4	100
Square_Root_conv ( )		☐			1	100
Pointer_Arithmetic ( )	1	☐	1		2	75
✓ IDP.6		☐			1	
! IDP.11	1	☐				
? IDP.21		☐	1			
✓ IDP.31		☐			1	
Close_To_Zero ( )		☐				0
Non_Infinite_Loop ( )		☐				0
RTE ( )		☐				0
Recursion_caller ( )		☐				0
Square_Root ( )		☐				0
Unreachable_Code ( )		☐				0
get_oil_pressure ( )		☐				0
__polyspace__stdstubs.c		☐				0
__polyspace_main.c		☐				0

Variables View: Written by, Read by, Written by task, Read by task, Potentially Written by, Potentially Read by

Variables: New_Project, __polyspace__stdstubs.errno

Call Tree View: ○ Both, ● Called by, ○ Calls, ☐ Complete, ☑ Update on selection

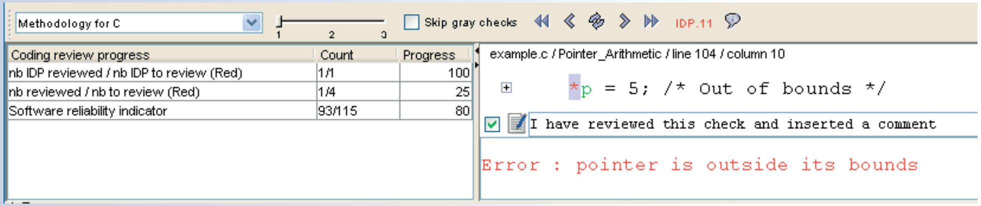
Calls: example.Pointer_Arithmetic, example.RTE

```
97              p++;
98          }
99
100     if(get_bus_status() > 0)
101         {
102             if(get_oil_pressure() > 0)
103                 {
104                     *p = 5; /* Out of bounds */
105                 }
106             else
107                 {
108                     i++;
109                 }
110         }
```

New_Project   Source file: example.c   example.c   Line: 1   Column: unknown

The methodological dashboard gives details and allows reviewing the check.

➤ Tick the review checkbox and type a comment in the text box on the right as follows:



Methodology for C | Skip gray checks | IDP.11

Coding review progress	Count	Progress
nb IDP reviewed / nb IDP to review (Red)	1/1	100
nb reviewed / nb to review (Red)	1/4	25
Software reliability indicator	93/115	80

example.c / Pointer_Arithmetic / line 104 / column 10

```
*p = 5; /* Out of bounds */
```

I have reviewed this check and inserted a comment

```
Error : pointer is outside its bounds
```

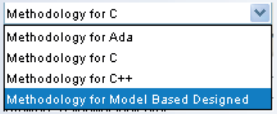
The left part of the dashboard has been updated, and displays some statistics in three lines:

- The first line gives the number and percentage of remaining checks to review in the selected category (here, red IDP checks).
- The second line gives values for the whole colour category (red, grey and unproven).
- The last line gives values for the whole software being reviewed.
  This is called `Software reliability indicator`. It gives the percentage of green checks compared to the total number of checks.

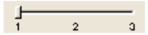
Other buttons in the Methodological dashboard allow navigating to the next ⏩ or previous ⏪ check that hasn't been reviewed yet. It's also possible to refresh the different views to come back to the check currently being reviewed using the ⟳ button.

3.4.2. Choose a methodological assistant

Some methodologies

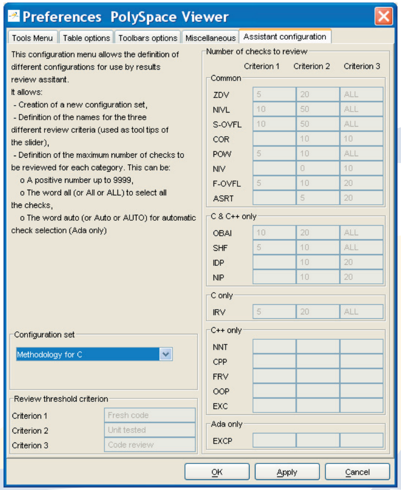


and associated levels



have been pre-defined by PolySpace.
The methodology allows selecting the categories of checks to review, the number for each category and their order based on statistics on many analysis results.
The level defines the number of checks to review by category. It is chosen according to the development phase during which the code has been analyzed: "`Fresh code`", "`Unit test`" and "`Code review`"
It is possible to define your own Methodology in the "`Preferences >PolySpace Viewer>Assistant methodology`" tab that is accessible from the "`Edit`" menu.

Here, you can create a new configuration set and define for each level what will be the categories of check to review and how many of each category.



This configuration menu allows the definition of different configurations for use by results review assitant.
It allows:
- Creation of a new configuration set,
- Definition of the names for the three different review criteria (used as tool tips of the slider),
- Definition of the maximum number of checks to be reviewed for each category. This can be:
  - o A positive number up to 9999,
  - o The word all (or All or ALL) to select all the checks,
  - o The word auto (or Auto or AUTO) for automatic check selection (Ada only)

Configuration set: Methodology for C

Review threshold criterion

Criterion 1	Fresh code
Criterion 2	Unit tested
Criterion 3	Code review

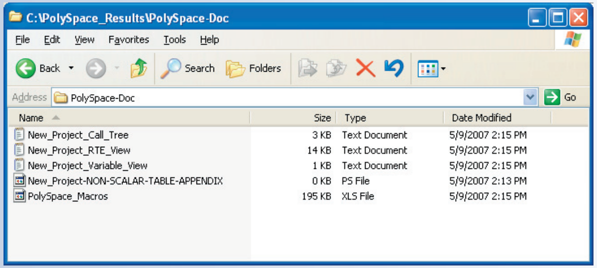
Number of checks to review

	Criterion 1	Criterion 2	Criterion 3
Common			
ZDV	5	20	ALL
NIVL	10	50	ALL
S-OVFL	10	50	ALL
COR		10	10
POW	5	10	ALL
NIV		0	10
F-OVFL	5	10	20
ASRT		5	20
C & C++ only			
OBAI	10	20	ALL
SHF	5	10	ALL
IDP		10	20
NIP		10	20
C only			
IRV	5	20	ALL
C++ only			
NNT			
CPP			
FRV			
OOP			
EXC			
Ada only			
EXCP			

3.5. Report Generation

When PolySpace performs an analysis, it generates textual files that can be used to create Excel® reports. These files are located in the results directory (See "`C:\PolySpace_Results\PolySpace-Doc`" or "`<PolySpaceInstallDir>\Examples\Demo_C\PolySpace-Doc`"). These files contain data related to all views except the source code one.

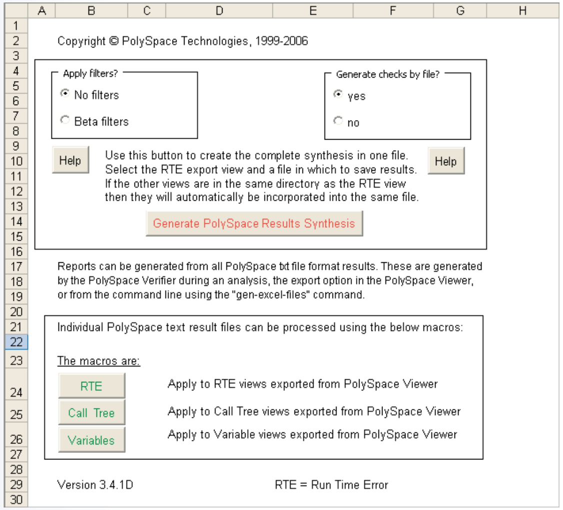
The "`C:\PolySpace_Results\PolySpace-Doc`" directory should contains the following files:


C:\PolySpace_Results\PolySpace-Doc

Address: PolySpace-Doc

Name	Size	Type	Date Modified
New_Project_Call_Tree	3 KB	Text Document	5/9/2007 2:15 PM
New_Project_RTE_View	14 KB	Text Document	5/9/2007 2:15 PM
New_Project_Variable_View	1 KB	Text Document	5/9/2007 2:15 PM
New_Project-NON-SCALAR-TABLE-APPENDIX	0 KB	PS File	5/9/2007 2:13 PM
PolySpace_Macros	195 KB	XLS File	5/9/2007 2:15 PM

➤ Open the file called "`PolySpace_Macros.xls`" and enable macros to display the Excel® file:


Copyright © PolySpace Technologies, 1999-2006

Apply filters? [✓ No filters] Beta filters

Generate checks by file? [✓ yes] no

Help

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.

Help

Generate PolySpace Results Synthesis

Reports can be generated from all PolySpace txt file format results. These are generated by the PolySpace Verifier during an analysis, the export option in the PolySpace Viewer, or from the command line using the "gen-excel-files" command.

Individual PolySpace text result files can be processed using the below macros:

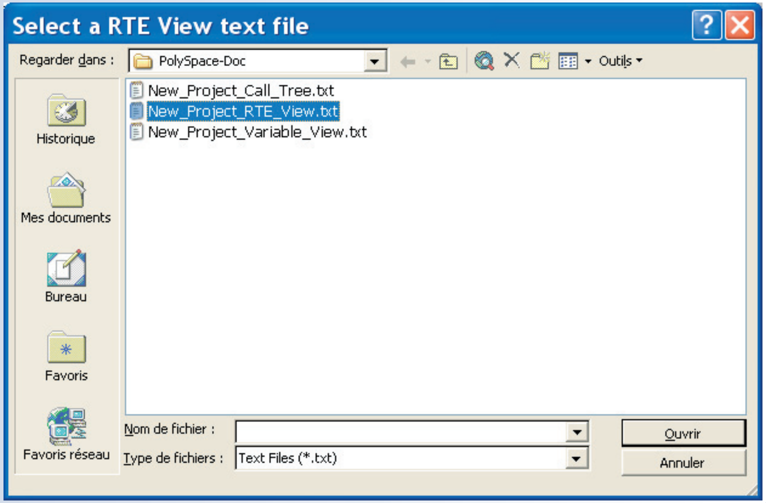
The macros are:

RTE	Apply to RTE views exported from PolySpace Viewer
Call Tree	Apply to Call Tree views exported from PolySpace Viewer
Variables	Apply to Variable views exported from PolySpace Viewer

Version 3.4.1D

RTE = Run Time Error

➤ Click on Generate PolySpace Results Synthesis A file browser opens. Select the file called "New_Project_RTE_View.txt" as shown below:


Select a RTE View text file

Regarder dans : PolySpace-Doc

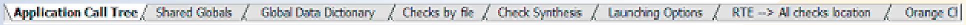
Outils

New_Project_Call_Tree.txt
New_Project_RTE_View.txt
New_Project_Variable_View.txt

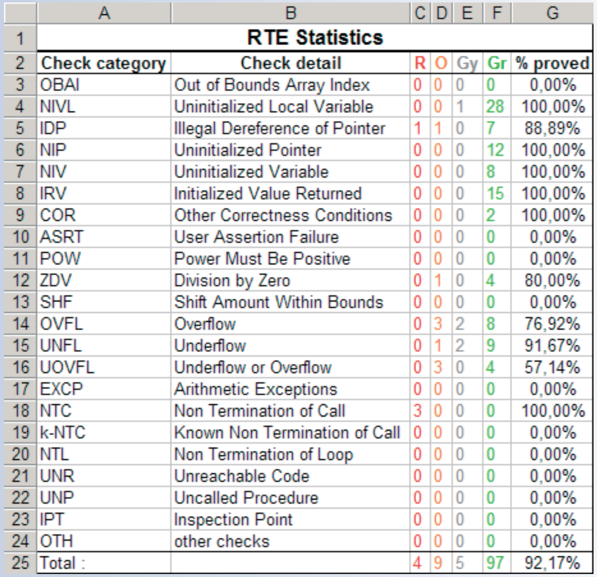
Nom de fichier :

Type de fichiers : Text Files (*.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 Cl

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%

4. Step 3:

Setting up and launching the MISRA-C checker

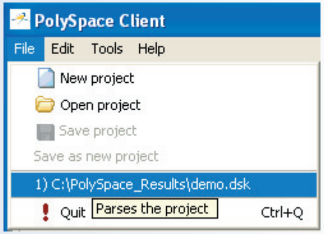
This paragraph describes how to perform the MISRA-C compliance verification in the analysis of "`example.c`". This verification takes place during the ANSI C compliance phase of the analysis.

4.1. Prerequisites

If PolySpace Client is already opened with the project defined previously (see Step 1 of this Getting Started) you can skip the following two steps.

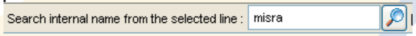
➤ If PolySpace Client has been closed, please open it again by double-clicking on the PolySpace Client icon:

➤ Select the saved project:



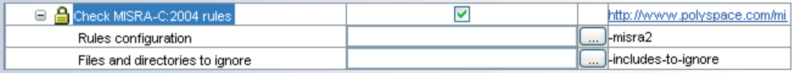
The project needs to be updated as follow:

➤ The "`Check MISRA rules`" option needs to be activated. To do so, type "`misra`" in the "`Search internal name from the selected line`" box



Step 3: Setting up and launching the MISRA-C checker

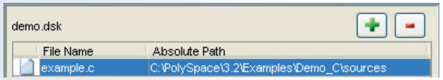
- Then click on 🔍. It will show all options containing the word "`misra`".
  Tick the "`Check MISRA-C: 2004 rules`" option and expand it to see the two associated sub-options `-misra2` and `-includes-to-ignore` :



⊟ Check MISRA-C:2004 rules	✓		http://www.polyspace.com/mi
Rules configuration		...	-misra2
Files and directories to ignore		...	-includes-to-ignore

We will detail theses two sub-options later.

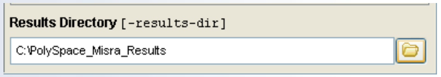
- Make sure "`example.c`" is selected:



demo.dsk

File Name	Absolute Path
example.c	C:\PolySpace\3.2\Examples\Demo_C\sources

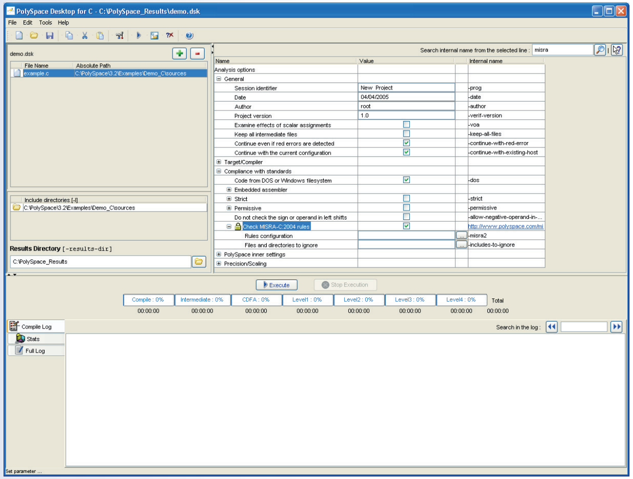
- Modify the results directory:



Results Directory `[-results-dir]`

C:\PolySpace_Misra_Results

The PolySpace Launcher should now look like that:



PolySpace Desktop for C - C:\PolySpace_Results\demo.dsk

demo.dsk

File Name	Absolute Path
example.c	C:\PolySpace\3.2\Examples\Demo_C\sources

Include directories [-I]

C:\PolySpace\3.2\Examples\Demo_C\sources

Results Directory `[-results-dir]`

C:\PolySpace_Results

Search internal name from the selected line : misra

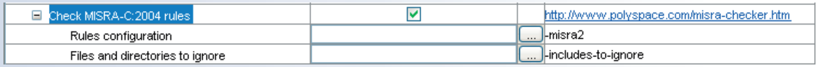
Name	Value		Internal name
Analysis options			
⊟ General			
Session identifier	New Project		-prog
Date	04/04/2005		-date
Author	root		-author
Project version	1.0		-verif-version
Examine effects of scalar assignments	☐		-voa
Keep all intermediate files	☐		-keep-all-files
Continue even if red errors are detected	✓		-continue-with-red-error
Continue with the current configuration	✓		-continue-with-existing-host
⊞ Target/Compiler			
⊟ Compliance with standards			
Code from DOS or Windows filesystem	✓		-dos
⊞ Embedded assembler			
⊞ Strict	☐		-strict
⊞ Permissive	☐		-permissive
Do not check the sign or operand in left shifts	☐		-allow-negative-operand-in-...
⊟ Check MISRA-C:2004 rules	✓		http://www.polyspace.com/mi
Rules configuration		...	-misra2
Files and directories to ignore		...	-includes-to-ignore
⊞ PolySpace inner settings			
⊞ Precision/Scaling			

Compile : 0%	Intermediate : 0%	CDFA : 0%	Level1 : 0%	Level2 : 0%	Level3 : 0%	Level4 : 0%	Total
00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00

4.2. Configuring PolySpace MISRA Checker

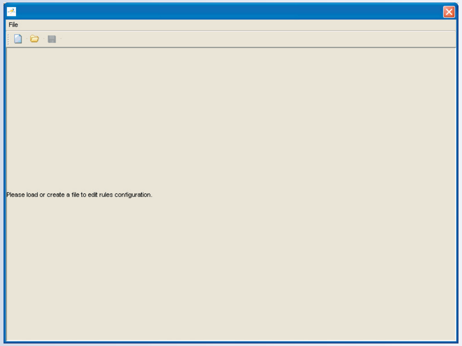
4.2.1. Selecting rules to verify

➤ Click on ... next to "-misra2".



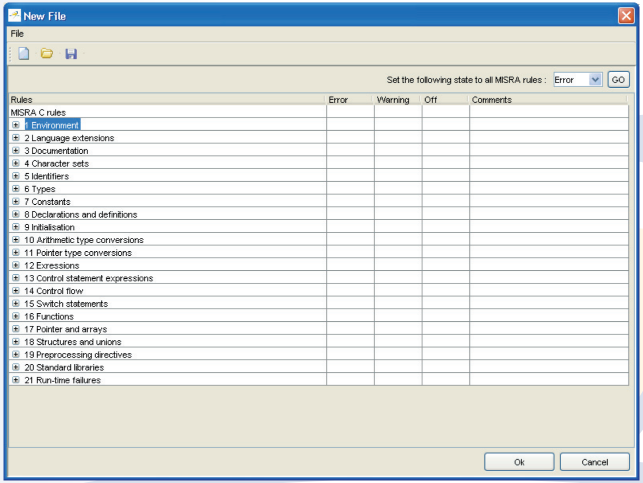
⊟ Check MISRA-C:2004 rules	✓		http://www.polyspace.com/misra-checker.htm
Rules configuration		...	-misra2
Files and directories to ignore		...	-includes-to-ignore

This opens a new window as below:



Please load or create a file to edit rules configuration.

➤ Click on . It creates a new MISRA-C configuration file. The content of the window is updated as below:

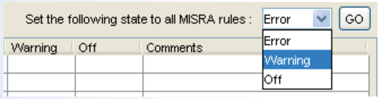


Rules	Error	Warning	Off	Comments
MISRA C rules				
⊞ 1 Environment				
⊞ 2 Language extensions				
⊞ 3 Documentation				
⊞ 4 Character sets				
⊞ 5 Identifiers				
⊞ 6 Types				
⊞ 7 Constants				
⊞ 8 Declarations and definitions				
⊞ 9 Initialisation				
⊞ 10 Arithmetic type conversions				
⊞ 11 Pointer type conversions				
⊞ 12 Exressions				
⊞ 13 Control statement expressions				
⊞ 14 Control flow				
⊞ 15 Switch statements				
⊞ 16 Functions				
⊞ 17 Pointer and arrays				
⊞ 18 Structures and unions				
⊞ 19 Preprocessing directives				
⊞ 20 Standard libraries				
⊞ 21 Run-time failures				

Each rule can be set as follows:

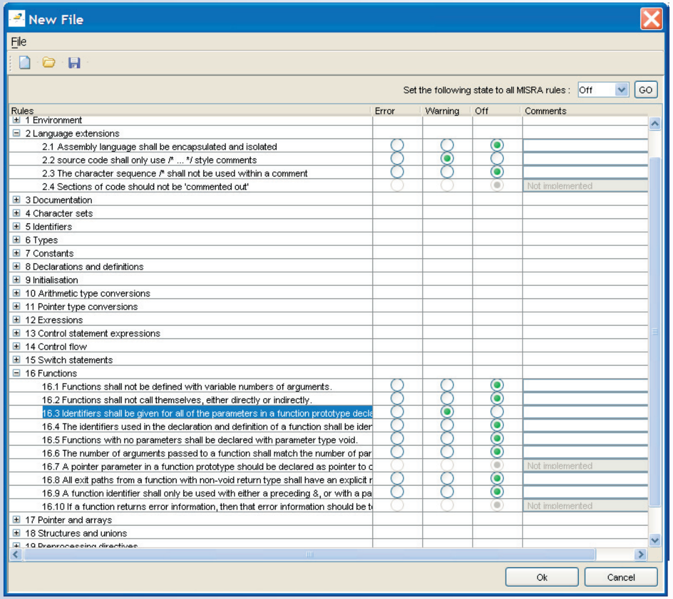
- "`Error`": this MISRA-C rule must be respected. If one or several deviations are detected, the analysis will stop at the end of the compilation phase.
- "`Warning`": if this MISRA-C rule is not respected, a warning will be displayed, but the analysis will continue.
- "`Off`": the MISRA-C rule will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

The default setting for all rules is “`Warning`”.


Set the following state to all MISRA rules : Error GO

➤ For the MISRA-C compliance verification of the "`example.c`" file, please update the setting to "`Off`" for all rules and apply it using the GO button

➤ Then, click on ⊞ to expand the set of rules 2 and 16. - "`Languages extensions`" and "`Functions`". The status of some of the underlying rules cannot be modified (rules 2.4, 16.7 and 16.10). Other rules have been turned to "`Off`". Click on the "`Warning`" column for rule 2.2 and on the "`Error`" column for rule 16.3. The green dot moves from column "`Off`" to column "`Warning`" and "`Error`"-. This means that PolySpace MISRA Checker will verify whether the rule 2.2 and 16.3 are respected and will stop after the ANSI compliance checking phase if rule 16.3 is not respected.

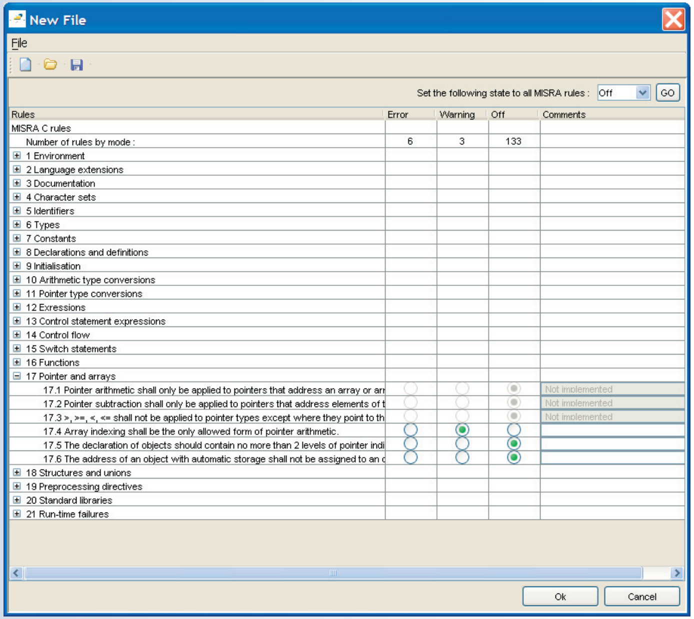

New File

File

Set the following state to all MISRA rules : Off GO

Rules	Error	Warning	Off	Comments
⊞ 1 Environment				
⊟ 2 Language extensions				
2.1 Assembly language shall be encapsulated and isolated	○	○	●	
2.2 source code shall only use /* ... */ style comments	○	●	○	
2.3 The character sequence /* shall not be used within a comment	○	○	●	
2.4 Sections of code should not be 'commented out'	○	○	●	Not implemented
⊞ 3 Documentation				
⊞ 4 Character sets				
⊞ 5 Identifiers				
⊞ 6 Types				
⊞ 7 Constants				
⊞ 8 Declarations and definitions				
⊞ 9 Initialisation				
⊞ 10 Arithmetic type conversions				
⊞ 11 Pointer type conversions				
⊞ 12 Exressions				
⊞ 13 Control statement expressions				
⊞ 14 Control flow				
⊞ 15 Switch statements				
⊟ 16 Functions				
16.1 Functions shall not be defined with variable numbers of arguments.	○	○	●	
16.2 Functions shall not call themselves, either directly or indirectly.	○	○	●	
16.3 Identifiers shall be given for all of the parameters in a function prototype decla	○	●	○	
16.4 The identifiers used in the declaration and definition of a function shall be ider	○	○	●	
16.5 Functions with no parameters shall be declared with parameter type void.	○	○	●	
16.6 The number of arguments passed to a function shall match the number of par	○	○	●	
16.7 A pointer parameter in a function prototype should be declared as pointer to c	○	○	●	Not implemented
16.8 All exit paths from a function with non-void return type shall have an explicit r	○	○	●	
16.9 A function identifier shall only be used with either a preceding &, or with a pa	○	○	●	
16.10 If a function returns error information, then that error information should be t	○	○	●	Not implemented
⊞ 17 Pointer and arrays				
⊞ 18 Structures and unions				
⊞ 19 Preprocessing directives				

➤ Click on ⊞ to expand the set of rules **17** - “Pointers and arrays” - and select “`Warning`” for rule **17.4.** This means that PolySpace MISRA Checker will verify whether the rule 17.4 is respected and display a warning message if this is not the case - but not stop the analysis.



Set the following state to all MISRA rules : Off GO

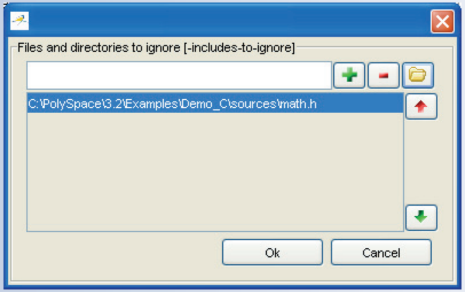
Rules	Error	Warning	Off	Comments
MISRA C rules				
Number of rules by mode :	6	3	133	
⊞ 1 Environment				
⊞ 2 Language extensions				
⊞ 3 Documentation				
⊞ 4 Character sets				
⊞ 5 Identifiers				
⊞ 6 Types				
⊞ 7 Constants				
⊞ 8 Declarations and definitions				
⊞ 9 Initialisation				
⊞ 10 Arithmetic type conversions				
⊞ 11 Pointer type conversions				
⊞ 12 Exressions				
⊞ 13 Control statement expressions				
⊞ 14 Control flow				
⊞ 15 Switch statements				
⊞ 16 Functions				
⊟ 17 Pointer and arrays				
17.1 Pointer arithmetic shall only be applied to pointers that address an array or arr	○	○	●	Not implemented
17.2 Pointer subtraction shall only be applied to pointers that address elements of t	○	○	●	Not implemented
17.3 >, >=, <, <= shall not be applied to pointer types except where they point to th	○	○	●	Not implemented
17.4 Array indexing shall be the only allowed form of pointer arithmetic.	○	●	○	
17.5 The declaration of objects should contain no more than 2 levels of pointer indi	○	○	●	
17.6 The address of an object with automatic storage shall not be assigned to an c	○	○	●	
⊞ 18 Structures and unions				
⊞ 19 Preprocessing directives				
⊞ 20 Standard libraries				
⊞ 21 Run-time failures				

➤ Click on OK. A “`Save As ...`” window opens, enabling to save the current configuration. Name it "`misrarules.txt`" and save it in the "`C:\MISRA_results`" directory.

4.2.2. 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 do so, click on ... 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.

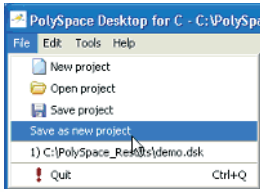

Files and directories to ignore [-includes-to-ignore]

C:\PolySpace\3.2\Examples\Demo_C\sources\math.h

➤ Select "`math.h`" using the browse button and click on OK to close the window. This file will not be checked.

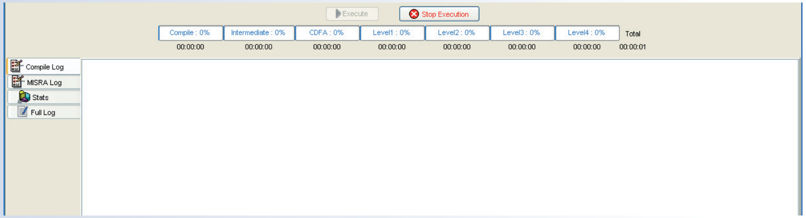
4.3. Running the MISRA checker

➤ 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.

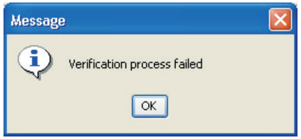

PolySpace Desktop for C - C:\PolySpa

➤Click on Execute to start the analysis.

During the new analysis a new filter button is displayed - MISRA Log - in the log view.
This button allows filtering out messages associated to MISRA-C compliance verification.



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



Verification process failed

➤ Click on 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.

If we focus on the MISRA log using the appropriate filter MISRA Log, only messages associated to MISRA-C are displayed.
The "Search in log" box allows searching for a specific keyword such as the reference number of a given rule. Let's search for "16.3".

```
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.
```

➤ 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 of 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.

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 a 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. Launch PolySpace Remotely

This paragraph describes the basic steps to launch an analysis in remote. To do so you need:

1. A Queue Manager server (QM) installed.
2. Your desktop PC configured with a PolySpace Client.
3. A networked machine configured with a PolySpace Server.

Please see the PolySpace Installation guide (available on the PolySpace CD-ROM in `\Docs\Install`) to install and configure, the Queue Manager, a Client and a Server.

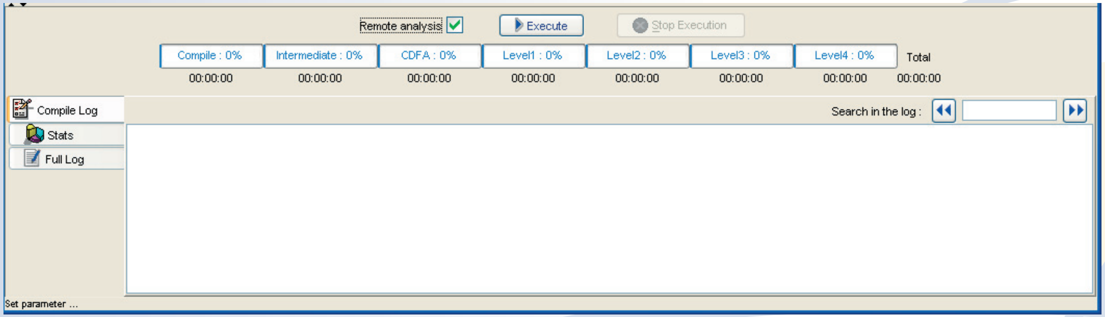
Note: Launching an analysis remotely requires a PolySpace Server product and associated license.

5.1. Launching an analysis

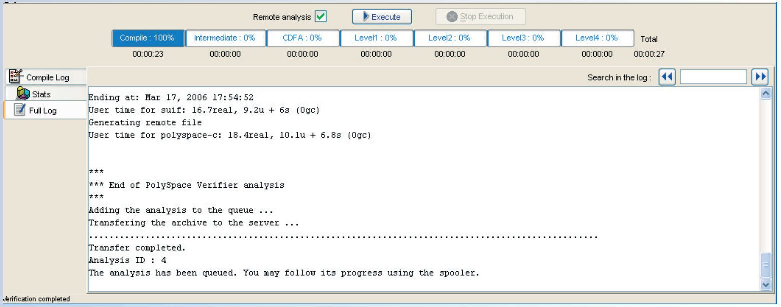
It can be done in two steps:

➤ Step 1: set up an analysis as described in Chapter 2 but do not launch it.

➤ Step 2: tick the "`Remote analysis`" checkbox (see figure below) and click on Execute 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. By clicking on the “Full Log” tab, you will see a message like this:

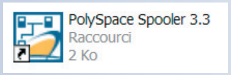


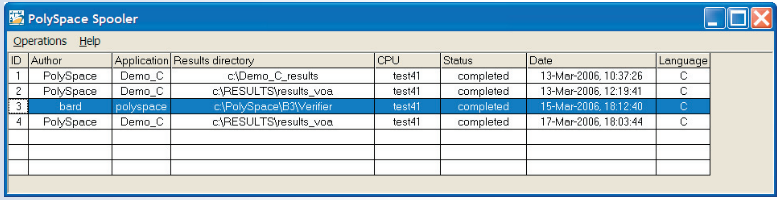
```
Ending at: Mar 17, 2006 17:54:52
User time for suif: 16.7real, 9.2u + 6s (0gc)
Generating remote file
User time for polyspace-c: 18.4real, 10.1u + 6.8s (0gc)


***
*** End of PolySpace Verifier analysis
***
Adding the analysis to the queue ...
Transfering the archive to the server ...
.................................................................................................
Transfer completed.
Analysis ID : 4
The analysis has been queued. You may follow its progress using the spooler.
```

The analysis has been queued with an ID number, and you can follow its progression using the PolySpace Spooler. If you do not tick the “`Remote analysis`” checkbox, the analysis continues locally.

5.2. Management of PolySpace analysis in remote: the PolySpace Spooler

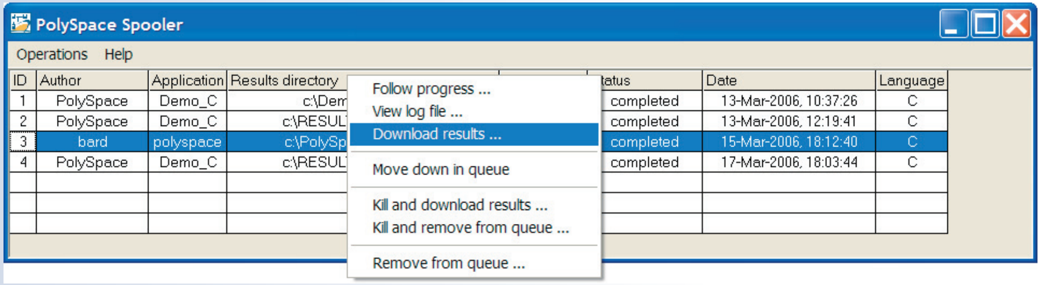
You can check the analysis processes in the queue by clicking on the short cut on your desktop PC or on the icon in the menu tab of the launcher.





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	bard	polyspace	c:\PolySpace\B3\Verifier	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

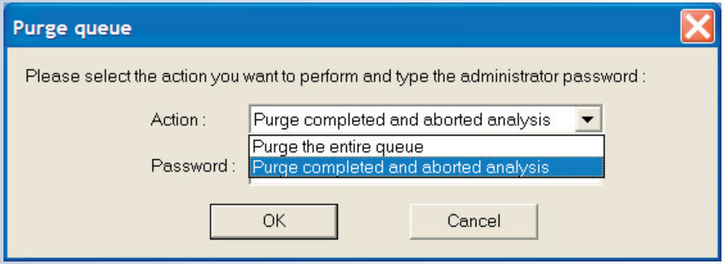
When you select an analysis and right click, you can manage it in the queue:



ID	Author	Application	Results directory	tatus	Date	Language
1	PolySpace	Demo_C	c:\Dem	completed	13-Mar-2006, 10:37:26	C
2	PolySpace	Demo_C	c:\RESUL	completed	13-Mar-2006, 12:19:41	C
3	bard	polyspace	c:\PolySp	completed	15-Mar-2006, 18:12:40	C
4	PolySpace	Demo_C	c:\RESUL	completed	17-Mar-2006, 18:03:44	C

- "*Follow progress*" displays 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.
- "*View log file*" displays the associated log file in a "Command prompt" window, in which you can see the last 100 updated lines of the log file in real time. This option is only available when the analysis is running.
- "*Download results*" downloads the results of an analysis to the Client. If the analysis is still running, already available partial results are downloaded on the Client, without disturbing the analysis. This option is not available for a "`queued`" analysis (that has not yet began).
- "*Move down in queue*" reduces the priority of a "`queued`" analysis.
- "*Kill and download results*" stops the analysis definitively and the latest available partial results are downloaded. The status of the analysis changes from "`running`" to "`aborted`". The analysis remains in the queue.
- "*Kill and remove from queue*" stops the analysis definitively and removes it from the queue. **The results will be lost.**
- "*Remove from queue*" removes a "`queued`", "`aborted`" or "`completed`" analysis. **The results will be lost.**

The queue can also be managed via the "`Operations>`" menu:


Purge queue

Please select the action you want to perform and type the administrator password :

Action : Purge completed and aborted analysis

Password :

• "`Operations>Purge queue`" purges the entire queue or purges only completed and aborted analysis (see below). The queue manager administrator password is required.

• "`Operations>Change root password`" changes administrator password of the queue manager. By default, the password is "`administrator`".

5.3. Batch commands

• **Launch analysis in batch:**
A set of commands allow the launching of analyses in batch (under a Cygwin shell on a Windows machine). All commands begin with the prefix `<PolySpaceCommonDir>/RemoteLauncher/bin/polyspace-remote-`. Commands available are polyspace-remote-c and polyspace-remote-desktop-c.

They are equivalent to the commands with a prefix `<PolySpaceInstallDir>/bin/polyspace-`.
For example, `polyspace-remote-desktop-c -server [<hostname>:[<port>] | auto]` allows the sending of a C Client analysis remotely.

- **Manage analysis in batch:**
  In batch and on a UNIX platform, a set of commands allow the management of analysis in the queue.
  All theses commands begin with the prefix `<PolySpaceCommonDir>/RemoteLauncher/bin/psqueue-`:
  - `psqueue-download <ID> <results dir>`: downloads an identified analysis into a results directory. `[-f]` forces download (without interactivity) and `-admin -p <password>` allows administrator to download results. Use `[-server <name>[:port]]` to select a specific Queue Manager. Use `[-v|version]` to indicate release number.
  - `psqueue-kill <ID>`: kills an identified analysis.
  - `psqueue-purge all|ended`: removes all or finished analyses in the queue.
  - `psqueue-dump`: gives the list of all analyses in the queue associated to the default Queue Manager.
  - `psqueue-move-down <ID>`: moves down an identified analysis in the queue.
  - `psqueue-remove <id>`: removes an identified analysis in the queue.
  - `psqueue-get-qm-server`: gives the name of the default Queue Manager.
  - `psqueue-progress <ID>`: gives progression of the currently identified and running analysis. `[-open-launcher]` displays the log in PolySpace launcher graphical user interface. `[-full]` gives full log file.
  - `psqueue-set-password <old password> <new password>`: changes administrator password.
  - `psqueue-check-config`: checks the configuration of Queue Manager. `[-check-licenses]` checks for licenses only.
  - `psqueue-upgrade`: allows upgrading a Client (see `PolySpace Install Guide` in the `<PolySpaceCommonDir>/Docs` directory). `[-list-versions]` gives the list of available releases for upgrade.
    `[-install-version <version number> [-install-dir <directory>]] [-silent]`
    allow to install an upgrade in a given directory potentially in silent mode.

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

5.4. Share analysis between accounts

• `analysis-key.txt` **file**

For security reasons, all analyses spooled are owned by the user who sent them. Each analysis has a unique crypted key.

The public part of the key is stored in a file named `analysis-keys.txt` and associated to a user account.
On a UNIX account, this file is located in: "`/home/<username>/.PolySpace`".On a Windows account,
it is located in: "`C:\Documents and Settings\<username>\Application Data\PolySpace`".

The format of the ASCII file is the following (`^t means` tabulation):
`<ID of launching> ^t <server name of IP address> ^t <public key>`

Example

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

When attempting to manage (download, kill and remove, etc.) a particular analysis, the Queue Manager will examine this file and check the associated public key before allowing the action.
If the key does not exist, an error message appears: "`key for analysis <ID> not found`".

So, if user A wants to manage (for example download results of) an analysis sent by user B, user A should edit his own `analysis-key.txt` file and add into it the line corresponding to that analysis in the `analysis-key.txt` file of user B.

• **Sharing analyses between projects with a magic key**

A magic key allows sharing analyses without taking into account the `<ID>`. It allows having the same key for all analyses launched. The format is the following: `0       <Server id>    <your hexadecimal value>`

All analyses spooled will use this key instead of a random one. This would allow any user that has this key in his `analysis-key.txt` file to manage all analyses sent with the magic key.

Note: The magic key only works for analyses launched after it has been set up.
Analyses sent before, will keep their auto-generated random keys.

6. Conclusion

After having followed each steps of this tutorial, you are now able to launch an analysis using PolySpace Client, enabling or not the MISRA verification phase and review some results with PolySpace Viewer.
All theses activities can be performed locally on your desktop PC or in a Client/Server architecture.

You will find more information on advanced options in "PolySpace C documentation.pdf" which is available on the PolySpace CD-ROM or by clicking on ? in PolySpace tools.

PolySpace
TECHNOLOGIES