Polyspace® Release Notes

Contents

Summary by Version

This table provides quick access to what's new in each version. For clarification, see "Using Release Notes" on page 6.

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Latest Version for C/C++ Products: V8.1 (R2011a)	Yes Details	Yes Summary	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports
Latest Version for Ada Products: V6.1 (R2011a)	Yes Details	Yes Summary	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports
Latest Version for Model Link Products: V5.7 (R2011a)	Yes Details	Yes Summary	Includes fixes: Polyspace Model Link SL Bug Reports Polyspace Model Link TL Bug Reports Polyspace UML Link RH Bug Reports
V8.0 (R2010b) for C/C++ Products:	Yes Details	Yes Summary	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
V6.0 (R2010b) for Ada Products:	Yes Details	Yes Summary	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports
V5.6 (R2010b) for Model Link Products:	Yes Details	Yes Summary	Includes fixes: Polyspace Model Link SL Bug Reports Polyspace Model Link TL Bug Reports Polyspace UML Link RH Bug Reports
V7.2 (R2010a) for C/C++ Products	Yes Details	Yes Summary	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports
V5.5 (R2010a) for Ada and Model Link Products	Yes Details	Yes Summary	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports Polyspace Model Link SL Bug Reports Polyspace Model Link TL Bug Reports Polyspace UML Link RH Bug Reports

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
V7.1 (R2009b) for C/C++ Products	Yes Details	Yes Summary	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports
V5.4 (R2009b) for Ada and Model Link Products	Yes Details	Yes Summary	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports Polyspace Model Link SL Bug Reports Polyspace Model Link TL Bug Reports Polyspace UML Link RH Bug Reports
V7.0 (R2009a) for C/C++ Products	Yes Details	Yes Summary	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
V5.3 (R2009a) for Ada and Model Link Products	Yes Details	No	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports Polyspace Model Link SL Bug Reports Polyspace Model Link TL Bug Reports Polyspace UML Link RH Bug Reports
V6.0 (R2008b) for C/C++ Products	Yes Details	No	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports
V5.2 (R2008b) for Ada and Model Link Products	Yes Details	No	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports Polyspace Model Link SL Bug Reports

Version (Release)	New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
V5.1 (R2008a)	Yes Details	Yes Summary	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports Polyspace Model Link SL Bug Reports
Previous Versions			Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports Polyspace Server for Ada Bug Reports Polyspace Model Link SL Bug Reports

Using Release Notes

Use release notes when upgrading to a newer version to learn about:

- New features
- Changes
- Potential impact on your existing files and practices

Review the release notes for other MathWorks® products required for this product (for example, MATLAB® or Simulink®). Determine if enhancements, bugs, or compatibility considerations in other products impact you.

If you are upgrading from a software version other than the most recent one, review the current release notes and all interim versions. For example, when you upgrade from V1.0 to V1.2, review the release notes for V1.1 and V1.2.

What Is in the Release Notes

New Features and Changes

- New functionality
- Changes to existing functionality

Version Compatibility Considerations

When a new feature or change introduces a reported incompatibility between versions, the **Compatibility Considerations** subsection explains the impact.

Compatibility issues reported after the product release appear under Bug Reports at the MathWorks Web site. Bug fixes can sometimes result in incompatibilities, so review the fixed bugs in Bug Reports for any compatibility impact.

Fixed Bugs and Known Problems

MathWorks offers a user-searchable Bug Reports database so you can view Bug Reports. The development team updates this database at release time and as more information becomes available. Bug Reports include provisions for any known workarounds or file replacements. Information is available for bugs existing in or fixed in Release 14SP2 or later. Information is not available for all bugs in earlier releases.

Access Bug Reports using your MathWorks Account.

Documentation on the MathWorks Web Site

Related documentation is available on `mathworks.com` for the latest release and for previous releases:

- Latest product documentation
- Archived documentation

Version 8.1 (R2011a) Polyspace for C/C++ Products

This table summarizes what's new in V8.1 (R2011a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for C/C++ Product" on page 8
- "Polyspace® Server for C/C++ Product" on page 24

Polyspace Client for C/C++ Product

Code Metrics (New for C++)

Code metric support, including cyclomatic number and other HIS metrics.

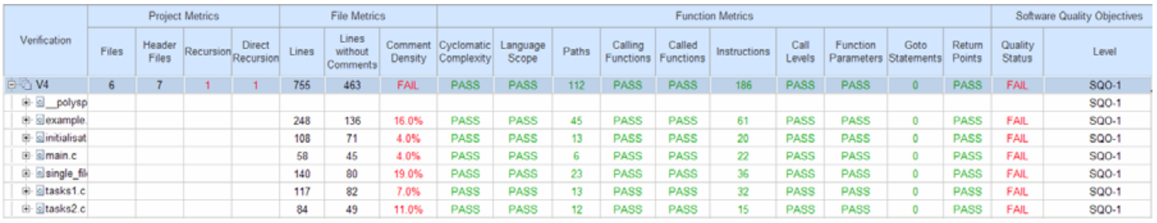
Polyspace® verification can now generate metrics about code complexity, which are based on the Hersteller Initiative Software (HIS) standard.

These metrics include:

- **Project metrics** – including number of recursions, number of include headers, and number of files.
- **File metrics** – including comment density, and number of lines.
- **Function metrics** – including cyclomatic number, number of static paths, number of calls, and Language scope.

When you run a verification with the `-code-metrics` option enabled, you can view software quality metrics data in the Polyspace Metrics Web interface

(**Code Metrics** view), or by running a Software Quality Objectives report from the Polyspace verification environment.



Verification	Project Metrics				File Metrics			Function Metrics											Software Quality Objectives	
	Files	Header Files	Recursion	Direct Recursion	Lines	Lines without Comments	Comment Density	Cyclomatic Complexity	Language Scope	Paths	Calling Functions	Called Functions	Instructions	Call Levels	Function Parameters	Goto Statements	Return Points	Quality Status	Level
V4	6	7	1	1	755	463	FAIL	PASS	PASS	112	PASS	PASS	186	PASS	PASS	0	PASS	FAIL	SQO-1
__polysp																			SQO-1
example.					248	136	16.0%	PASS	PASS	45	PASS	PASS	61	PASS	PASS	0	PASS	FAIL	SQO-1
initialisat					108	71	4.0%	PASS	PASS	13	PASS	PASS	20	PASS	PASS	0	PASS	FAIL	SQO-1
main.c					58	45	4.0%	PASS	PASS	6	PASS	PASS	22	PASS	PASS	0	PASS	FAIL	SQO-1
single_fil					140	80	19.0%	PASS	PASS	23	PASS	PASS	36	PASS	PASS	0	PASS	FAIL	SQO-1
tasks1.c					117	82	7.0%	PASS	PASS	13	PASS	PASS	32	PASS	PASS	0	PASS	FAIL	SQO-1
tasks2.c					84	49	11.0%	PASS	PASS	12	PASS	PASS	15	PASS	PASS	0	PASS	FAIL	SQO-1

The software generates numeric values or pass/fail results for various metrics.

For more information, see "Software Quality with Polyspace Metrics" in the *Polyspace Products for C++ User's Guide*.

Saving Polyspace Metrics Review

Previously, when you saved your project (**Ctrl+S**) after a review of results from Polyspace Metrics, the software would save your comments and justifications both locally and in the Polyspace Metrics repository.

Now, if you save your project (**Ctrl+S**), the software saves your review to a local folder only. A new button is available on the Run-Time Checks toolbar. If you click this button, the software saves your comments and justifications to a local folder *and* the Polyspace Metrics repository.

This feature allows you to upload your review to the repository only when you are satisfied that your review is, for example, correct and complete.

You can still configure your software to display the previous behavior.

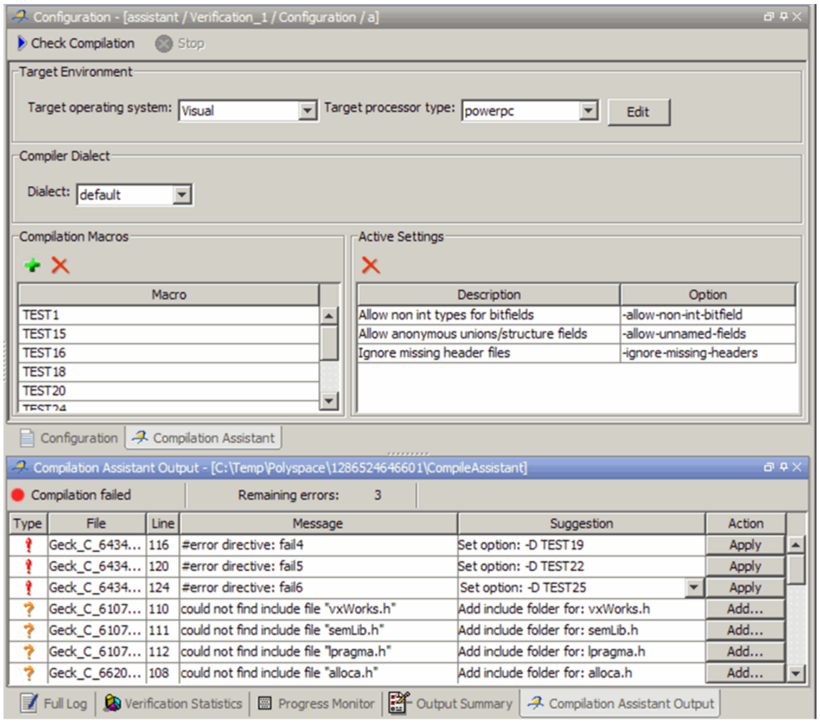
For more information, see "Saving Review Comments and Justifications" in the *Polyspace Products for C User's Guide*.

Compilation Assistant

New Compilation Assistant to ease project configuration (cross-compiler settings).

The Compilation Assistant allows you to check your project for compilation problems before launching a verification. The Compilation Assistant then:

- Automatically detects pre-processing, compilation, and dialect options required for your code (for example, `-I` and `-D`).
- Provides suggestions to solve compilation problems.



Target Environment

Target operating system: Visual

Target processor type: powerpc

Compiler Dialect

Dialect: default

Compilation Macros

Macro
TEST1
TEST15
TEST16
TEST18
TEST20
TEST24

Active Settings

Description	Option
Allow non int types for bitfields	-allow-non-int-bitfield
Allow anonymous unions/structure fields	-allow-unnamed-fields
Ignore missing header files	-ignore-missing-headers

Compilation Assistant Output - [C:\Temp\Polyspace\1286524646601\CompileAssistant]

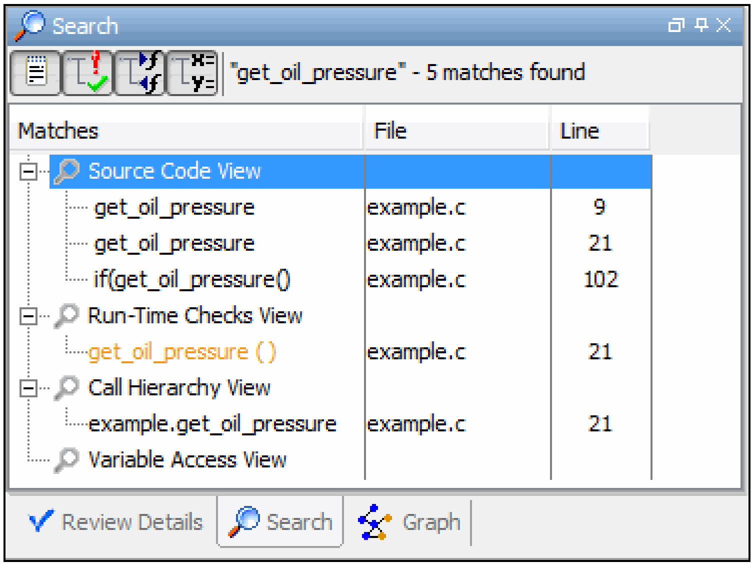
Compilation failed   Remaining errors: 3

Type	File	Line	Message	Suggestion	Action
!	Geck_C_6434...	116	#error directive: fail4	Set option: -D TEST19	Apply
!	Geck_C_6434...	120	#error directive: fail5	Set option: -D TEST22	Apply
!	Geck_C_6434...	124	#error directive: fail6	Set option: -D TEST25	Apply
?	Geck_C_6107...	110	could not find include file "vxWorks.h"	Add include folder for: vxWorks.h	Add...
?	Geck_C_6107...	111	could not find include file "semLib.h"	Add include folder for: semLib.h	Add...
?	Geck_C_6107...	112	could not find include file "lpragma.h"	Add include folder for: lpragma.h	Add...
?	Geck_C_6620...	108	could not find include file "alloca.h"	Add include folder for: alloca.h	Add...

For more information, see "Checking for Compilation Problems" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*

Improved Search Function

Enhanced search functionality in the Run-Time Checks perspective allows you to perform a search in several views at once (call hierarchy, variable access, run-time checks and source code), and provides search results in a single "Search" view.



"get_oil_pressure" - 5 matches found

Matches	File	Line
Source Code View		
get_oil_pressure	example.c	9
get_oil_pressure	example.c	21
if(get_oil_pressure()	example.c	102
Run-Time Checks View		
get_oil_pressure ( )	example.c	21
Call Hierarchy View		
example.get_oil_pressure	example.c	21
Variable Access View		

For more information, see "Searching Results in Run-Time Checks Perspective" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Back to Source Function in Run-Time Checks Perspective

Improved navigation from the Run-Time Checks perspective to the source code containing a check.

You can now right-click a check in your verification results, and open the source file containing that check.

You can configure the software to open source files in either a text editor, or your IDE.

For more information, see “Configuring Text and XML Editors” in the *Polyspace Products for C User’s Guide* or *Polyspace Products for C++ User’s Guide*.

Structure Fields in Data Dictionary

Distinction of variable fields in the Data Dictionary provides a more accurate Data Dictionary.

The enhanced Data Dictionary:

- Helps locate specific field accesses.
- Provides more information on fields (number of read/write accesses, field type).
- Provides a hierarchical view of structured variables.

For more information, see “Variable Access Pane” in the *Polyspace Products for C User’s Guide* or *Polyspace Products for C++ User’s Guide*.

Overflow Check Customization

New options allow you to customize how OVFL checks are handled during verification. You can customize computation through overflow constructions, control the presence of overflow checks, and the dynamic behavior in case of a run-time error.

These options allow you to:

- Not generate OVFL checks on all computations (values are computed the same way processors do).
- Not truncate the value after an OVFL check, and carry on with wrapped values (OVFL check does not impact values during verification).

For more information, see “Detect overflows on (`-scalar-overflows-checks`)” and “Overflows computation mode (`-scalar-overflows-behavior`)” in the *Polyspace Products for C Reference*.

Compatibility Considerations. The option `-detect-unsigned-overflows` (available in previous releases) has been renamed. To achieve the same behavior as the previous option, specify the new option `-scalar-overflows-checks signed-and-unsigned`.

When using the new options, your verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Main Generator Improvements

Enhanced main-generator to improve verification results for generated code.

The new main-generator allows you greater control over the behavior of the generated main. New options allow you to generate a main specifically designed for cyclic programs, to support generated code and Model-Based Design. This improves verification results at the subsystem level.

The generated main now has the following behavior:

1. It initializes any variables identified by the option `-variables written-before-loop`.
2. It calls any functions specified by the option `-functions-called-before-loop`. This could be considered an initialization function.
3. It initializes any variables identified by the option `-variables written-in-loop`.
4. It calls any functions specified by the option `-functions-called-in-loop`.
5. It calls any functions specified by the option `-functions-called-after-loop`. This could be a terminate function for a cyclic program.

For more information, see “Automatically Generating a Main” in the *Polyspace Products for C User’s Guide*.

Compatibility Considerations. Due to precision improvements, verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

In addition, several Analysis options have been renamed to support the new main generator.

Previous Name (R2010b)	New Name (R2011a)
`-main-generator-writes-variables`	`-variables-written-before-loop`
`-function-called-before-main`	`-function-called-before-loop`
`-main-generator-calls`	`-functions-called-in-loop`

If you have any scripts that use the old options, update them to reflect the new names.

Verification Time Limit

You can now specify a time limit for verifications using the `-timeout` option. If the verification does not complete within the specified time, the verification fails.

For more information, see "Verification time limit (`-timeout`)" in the *Polyspace Products for C Reference* or *Polyspace Products for C++ Reference*.

Continue Verification with Compile Errors

You can now specify that a verification continues even if some source files do not compile, using the option `-continue-with-compile-error`.

Functions that are used but not specified are stubbed automatically.

If a source file contains global variables, you may also need to select the option `-allow-undef-variables` to enable verification.

For more information, see "Continue with compile error (`-continue-with-compile-error`)" in the *Polyspace Products for C Reference* or *Polyspace Products for C++ Reference*.

Precision Improvements

Improved precision on arrays and functions, resulting in less orange checks.

The precision improvements effect:

- `NIV`, `NIVL`, `NIP`, and `IRV` checks
- array cells
- boolean decision graphs
- various other constructs

Compatibility Considerations. Verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Permissive Mode Set By Default

Permissive verification mode is now set by default for new projects. This reduces the number of compilation errors for verifications launched with default settings.

The following options are now set by default:

- `-discard-asm`
- `-allow-non-in-bitfields`
- `-permissive-link`
- `-allow-undef-variables`
- `-allow-unnamed-fields`
- `-allow-negative-operand-in-shift`
- `-allow-language-extensions`

If you want to use stricter compilation settings, you can select them in the project configuration.

Compatibility Considerations. When using the default options, your results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Default Project Location

On Windows systems, the default project location has changed.

The default project location is now in `My Documents`. Previously, the default location was defined in the user profile.

Variable Range Inconsistency between Variable Access Pane and Tooltips

The range given for a variable in the Variable Access Pane (Variables View) can differ from the range given by tooltips on the reads of a variable in the Source code view. The range provided by the tooltip will be wider than the range given in the Variables View.

This difference is due to imprecision in the tooltip. The Variables View provides the correct range for the variable.

For example:

- Variables View states that variable *X* is in range `[0..4000]`
- Tooltip on a read of *X* states that the range is `[0,7000]`.

In this case, `[0..4000]` is the correct range. The tooltip range is caused by imprecision that may be fixed in future releases.

Visual Studio Integration

New Visual Studio® import tool allows you to automatically extract some Polyspace settings from a Visual Studio project file.

This tool can help you:

- Locate source files, include folders and preprocessing directives
- Set some Polyspace Visual Studio specific options

For more information, see "Importing Visual Studio Project Information into Polyspace Project" in the *Polyspace Products for C++ User's Guide*.

Product Name Change in Files and Folders

The Polyspace product name has changed from "PolySpace" to "Polyspace" in R2011a. This change impacts the name of all files and folders created by the software.

For example:

- `PolySpace-Doc` folder has changed to `Polyspace-Doc`
- `PolySpace_xxxx.log` file has changed to `Polyspace_xxxx.log`

Compatibility Considerations. If you have existing folders that use the previous product name (for example, `PolySpace/PolySpace_Common`) the R2011a installation will continue to use these existing folders. However, any files or folders created during or after installation will use the new name.

If you have any shortcuts or scripts that are case-sensitive, you should update them to use the correct name.

Visual Studio Support

Added support for Microsoft® Visual Studio 2010.

For more information, see the *Polyspace Installation Guide*.

Eclipse IDE Support

Added support for Version 3.6 of the Eclipse IDE.

For more information, see the *Polyspace Installation Guide*.

License Manager Support

The License Manager for Polyspace products has been upgraded to FLEXnet® 11.9.

You may need to upgrade your FLEXnet server and daemon.

For more information, see “Polyspace License Installation” in the *Polyspace Installation Guide*.

Changes to Verification Results

- “Compatibility Considerations” on page 18
- “Certain COR Checks Changing to OVFL ” on page 18
- “COR Checks on Function Pointer” on page 19
- “NIV Check on Local Volatile Variables” on page 19
- “OVFL Checks on Assignment” on page 19
- “Precision Improvements for NIV Checks” on page 20
- “Precision Improvements on Arrays and Functions” on page 20
- “Compilation Errors for Classes without Constructors” on page 20

Compatibility Considerations. Verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Refer to the following sections for information on the specific changes.

Certain COR Checks Changing to OVFL . In previous releases, certain types of overflow errors were reported as COR checks instead of OVFL checks. For example:

```
typedef long int32;
extern int32 random(void);


int32 func(int32 a, int32 b)
{
int32 res = 0;
if (random()) {
res =  a/b;    // COR changing to OVFL in R2011a
}
return res;
}
```

These checks are now reported as OVFL, which will impact check statistics when compared to previous releases.

COR Checks on Function Pointer. In previous releases, verification reported a COR check on function pointer when the parameter type of function pointer is `void*`. For example:

```
typedef void (*func)(void*);

void foo(int *p) { *p=1; }

void bar(void)
{
int a;
func A = foo;
A(&a);
}
```

In R2011a, verification considers that type `void *` is compatible with all other pointer types.

This may result in changes to the color of `COR` checks. The Call graph may also been impacted. It can also have an impact on performance and precision (more calls considered).

NIV Check on Local Volatile Variables. The behavior of NIV checks for local volatile variables has changed.

In previous releases, the NIV for a local volatile variable was always orange. In R2011a, verification allows local volatile variables to behave just like other variables – if they are initialized in the code, the NIV is green.

Polyspace verification considers that the hardware can bring a value (so NIV cannot be red) but will not de-initialize. Therefore, if the variable is initialized by the code, it is green.

OVFL Checks on Assignment. By default, verification no longer reports `OVFL` checks on assignment, for example:

```
uc = ~uc;
```

The total number of checks in your results may change when compared to previous releases.

If you want the verification to report these types of checks, you can use the option `-detect-overflows-on-operator-not` to retain the previous behavior.

Precision Improvements for NIV Checks. Improved precision on `NIV`, `NIVL`, `NIP`, and `IRV` checks.

Precision Improvements on Arrays and Functions. Improved precision on arrays and functions.

Compilation Errors for Classes without Constructors. In previous releases, a compilation error occurs when you use the options `-unit-by-unit` or `-class-analyzer all` on source code containing classes with no user defined or compiler generated constructor.

In R2011a, this behavior changes as follows:

- No compilation error occurs.
- When using the options `-unit-by-unit` or `-class-analyzer all`, if a class has no constructor, all of its members are randomly initialized to the full range.
- When using the option `-class-analyzer custom-class-list`, if a class among the custom-class-list has no constructor, the verification does not initialize the class members in order to highlight NIV/NIP on accessing the class members (which mean that this class instance can never be correctly constructed).
- A warning is displayed in the log file.

Changes to Coding Rules Checker Results

- "Compatibility Considerations" on page 21
- "MISRA C Rule 12.1 – Parentheses for Operand of Unary Operator. " on page 21

- “Single Rule Violation Reported Multiple Times” on page 21

Compatibility Considerations. Due to changes in the coding rules checker, the number of coding rule violations may change when compared to previous versions of the software.

Refer to the following sections for information on the specific changes.

MISRA C® Rule 12.1 – Parentheses for Operand of Unary Operator.
. In previous releases, the coding rules checker could incorrectly report a violation of MIRSA C Rule 12.1 for the operand of a unary operator. For example:

```
Y1 =  (U1 * U2) -0.366;      // Passes 12.1
Y2 =  (-1 * (0.366)) + (U1 * U2);   // Fails 12.1
Y3 =  -0.366 + (U1 * U2);     // Fails 12.1
4 = 0.366 + (U1 * U2);      // Passes 12.1
```

The MISRA rule states that parentheses are not required for the operand of a unary operator.

The number of violations of Rule 12.1 may decrease when compared to previous releases.

Single Rule Violation Reported Multiple Times. In previous releases, Polyspace Metrics could report more than one violation of a single coding rule in the same location. This occurred when the message of a rule violation was modified, and the same results were uploaded to the Metrics database multiple times.

In R2011a, messages for rule violations that have the same ID and the same location are merged into a single message of only one rule violation

Therefore, the total number of rule violations may be lower in R2011a than in previous releases.

Changes to Analysis Options

New Options.

Option	For more information
Variables written in loop (`-variables-written-in-loop`)	"Main Generator Improvements" on page 13
Functions called after loop (`-functions-called-after-loop`)	"Main Generator Improvements" on page 13
Overflow computation mode (`-scalar overflows-behavior`)	"Overflow Check Customization" on page 12
Continue with compile error (`-continue-with-compile-error`)	"Continue Verification with Compile Errors" on page 14
Verification time limit (`-timeout`)	"Verification Time Limit" on page 14

Changes to Existing Options. The following options have been renamed in R2011a.

New Name (R2011a)	Previous Name (R2010b)	Change
Target operating system	**Operating system target for PolySpace stubs**	GUI name only
Ignore assembly code	**Discard Assembly code**	GUI name only
Allow non int types for bitfields	**Allow non-ANSI/ISO C-90 types of bitfields**	GUI name only
Allow undefined global variables	**Continue even with undefined global variables**	GUI name only
Ignore overflowing computations on constants	**Permits overflowing computations on constants**	GUI name only
Allow anonymous unions/structure fields	**Allow un-named Unions/Structures**	GUI name only
Allow negative operand for left shifts	**Do not check the sign of operand in left shifts**	GUI name only

New Name (R2011a)	Previous Name (R2010b)	Change
Ignore missing header files	**No error on missing header file**	GUI name only
Variables written before loop (`-variables-written-before-loop`)	**Write accesses to global variables** (`-main-generator-writes-variables`)	GUI and command-line name See "Main Generator Improvements" on page 13
Functions called before loop (`-functions-called-before-loop`)	**First functions to call** (`-function-called-before-main`)	GUI and command-line name See "Main Generator Improvements" on page 13
Functions called in loop (`-functions-called-in-loop`)	**Function calls** (`-main-generator-calls`)	GUI and command-line name See "Main Generator Improvements" on page 13
Detect overflows on (`-scalar-overflows-checks`)	**Detect overflows on unsigned integers** (`-detect-unsigned-overflows`)	Functionality change GUI and command line name See "Overflow Check Customization" on page 12

In addition, the default settings for some **Permissive** options have changed.

Deprecated Options. None.

Polyspace Server for C/C++ Product

Code Metrics (New for C++)

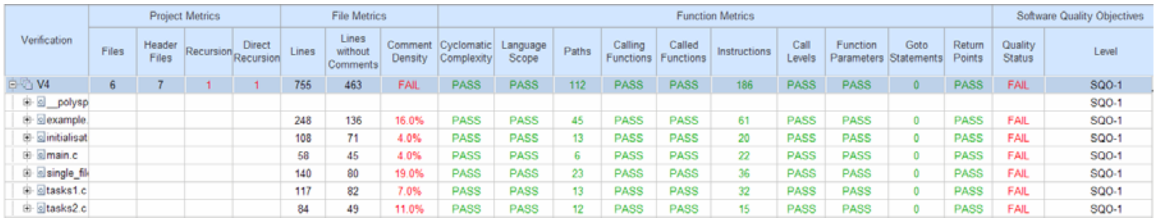
Code metric support, including cyclomatic number and other HIS metrics.

Polyspace verification can now generate metrics about code complexity, which are based on the Hersteller Initiative Software (HIS) standard.

These metrics include:

- **Project metrics** – including number of recursions, number of include headers, and number of files.
- **File metrics** – including comment density, and number of lines.
- **Function metrics** – including cyclomatic number, number of static paths, number of calls, and Language scope.

When you run a verification with the `-code-metrics` option enabled, you can view software quality metrics data in the Polyspace Metrics Web interface (**Code Metrics** view), or by running a Software Quality Objectives report from the Polyspace verification environment.



Verification	Project Metrics				File Metrics			Function Metrics											Software Quality Objectives	
	Files	Header Files	Recursion	Direct Recursion	Lines	Lines without Comments	Comment Density	Cyclomatic Complexity	Language Scope	Paths	Calling Functions	Called Functions	Instructions	Call Levels	Function Parameters	Goto Statements	Return Points	Quality Status	Level
V4	6	7	1	1	755	463	FAIL	PASS	PASS	112	PASS	PASS	186	PASS	PASS	0	PASS	FAIL	SQO-1
__polysp																			SQO-1
example.					248	136	16.0%	PASS	PASS	45	PASS	PASS	61	PASS	PASS	0	PASS	FAIL	SQO-1
initialisat					108	71	4.0%	PASS	PASS	13	PASS	PASS	20	PASS	PASS	0	PASS	FAIL	SQO-1
main.c					58	45	4.0%	PASS	PASS	6	PASS	PASS	22	PASS	PASS	0	PASS	FAIL	SQO-1
single_fil					140	80	19.0%	PASS	PASS	23	PASS	PASS	36	PASS	PASS	0	PASS	FAIL	SQO-1
tasks1.c					117	82	7.0%	PASS	PASS	13	PASS	PASS	32	PASS	PASS	0	PASS	FAIL	SQO-1
tasks2.c					84	49	11.0%	PASS	PASS	12	PASS	PASS	15	PASS	PASS	0	PASS	FAIL	SQO-1

The software generates numeric values or pass/fail results for various metrics.

For more information, see “Software Quality with Polyspace Metrics”in the *Polyspace Products for C++ User’s Guide.*

Saving Polyspace Metrics Review

Previously, when you saved your project (**Ctrl+S**) after a review of results from Polyspace Metrics, the software would save your comments and justifications both locally and in the Polyspace Metrics repository.

Now, if you save your project (**Ctrl+S**), the software saves your review to a local folder only. A new button is available on the Run-Time Checks toolbar. If you click this button, the software saves your comments and justifications to a local folder *and* the Polyspace Metrics repository.

This feature allows you to upload your review to the repository only when you are satisfied that your review is, for example, correct and complete.

You can still configure your software to display the previous behavior.

For more information, see "Saving Review Comments and Justifications" in the *Polyspace Products for C User's Guide*.

Automatic Comment Import for Server Verifications

When you download results from the Polyspace server, the software now automatically imports any comments from results in the destination folder into the downloaded results (except for verifications using the option `-add-to-results-repository`).

As a result of this change, you can now download intermediate results for a verification running on the Polyspace server, and add or edit comments on those results. When you later download the final results, your comments are preserved.

You can also download and comment on a single unit of a unit-by-unit verification, even if other units are still pending in the server queue. When you download the final results (which overwrites the earlier results), your comments are preserved.

License Manager Support

The License Manager for Polyspace products has been upgraded to FLEXnet 11.9.

You may need to upgrade your FLEXnet server and daemon.

For more information, see “Polyspace License Installation” in the *Polyspace Installation Guide*.

Version 6.1 (R2011a) Polyspace for Ada Products

This table summarizes what's new in V6.1 (R2011a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for Ada Product" on page 27
- "Polyspace® Server for Ada Product" on page 35

Polyspace Client for Ada Product

Code Metrics

Polyspace Metrics now generates Ada code metrics, giving you the number of:

- Protected shared variables
- Unprotected shared variables
- Files
- Lines of code
- Packages
- Packages that appear in `with` statements
- Subprograms that appear in `with` statements

You can view the metrics by:

- Using a Web browser to access the **Code Metrics** view of your project in Polyspace Metrics
- Examining the XML file that the software generates

For more information, see "Setting Up Verification to Generate Metrics" and "Review Code Metrics" in the *Polyspace Products for Ada User's Guide*.

Saving Polyspace Metrics Review

Previously, when you saved your project (**Ctrl+S**) after a review of results from Polyspace Metrics, the software would save your comments and justifications both locally and in the Polyspace Metrics repository.

Now, if you save your project (**Ctrl+S**), the software saves your review to a local folder only. A new button is available on the Run-Time Checks toolbar. If you click this button, the software saves your comments and justifications to a local folder *and* the Polyspace Metrics repository.

This feature allows you to upload your review to the repository only when you are satisfied that your review is, for example, correct and complete.

You can still configure your software to display the previous behavior.

For more information, see "Saving Review Comments and Justifications" in the *Polyspace Products for Ada User's Guide*.

Support for Rational and Aonix Compilers

The software now provides support for the IBM® Rational® Apex and Aonix® compilers. For more information, see "Operating system target for Standard Libraries compatibility" in the *Polyspace Products for Ada Reference*.

Multi-Core Support

On multi-core computers, you can reduce verification time by specifying the use of core processors simultaneously to perform the verification. The software provides a new command line option `max-processes`, which uses a default value of 4. You can specify any value between 1 and 128. For more information, see "Number of processes for multiple CPU core systems" in the *Polyspace Products for Ada Reference*.

Generated Main with Explicit Tasks and Accept Statements

If you select the option `Generate a main` and there are explicit tasks in the source code, then task bodies are verified like subprograms and `accept` statements are not executed. After verification, code associated with the `accept` statements is gray. For more information, see "Generate a main" in the *Polyspace Products for Ada Reference*.

Compatibility Considerations. Previously, if there were explicit tasks and `accept` statements in the source code, a verification run could have generated red, green, or orange checks for code within the `accept` statements. Now, verification colors the code within these `accept` statements gray.

Enhancements in Run-Time Checks Perspective

Within the source code view, you can investigate checks with tooltips that provide range information about variables.

In addition, when you click a check, the software provides information about the check in the Review Details pane.

For more information, see "Using Range Information in Run-Time Checks Perspective" and "Selecting a Check to Review" in the *Polyspace Products for Ada User's Guide*.

Compatibility Considerations . Because of these enhancements, the IPT and VOA checks are no longer required, and the software does not generate these checks anymore.

UOVFL and UNFL Checks Removed

The software no longer generates UOVFL and UNFL checks, but generates OVFL checks in place of these checks.

Compatibility Considerations. Due to the replacement of UOVFL and UNFL checks by the OVFL check, the number of green checks may decrease when compared with previous versions of the software. For example, a combination of an orange OVFL and a green UNFL generated by a previous version may be replaced by a single orange OVFL.

NIV Checks for Universal Constants

The software now generates NIV checks for read operations on universal constants. If the constants are used in your code, the NIV checks are green. If the constants are in unreacheable code, the NIV checks are gray.

Compatibility Considerations. This enhancement may change the check metrics and selectivity of the verification. The number of green and gray checks may be higher compared to the number generated by previous versions of the software.

Variable Range Inconsistency between Variable Access Pane and Tooltips

The range given for a variable in the Variable Access Pane (Variables View) can differ from the range given by tooltips on the reads of a variable in the Source code view. The range provided by the tooltip will be wider than the range given in the Variables View.

This difference is due to imprecision in the tooltip. The Variables View provides the correct range for the variable.

For example:

- Variables View states that variable *X* is in range `[0..4000]`
- Tooltip on a read of *X* states that the range is `[0,7000]`.

In this case, `[0..4000]` is the correct range. The tooltip range is caused by imprecision that may be fixed in future releases.

Verification Time Limit

You can now specify a time limit for verifications using the `-timeout` option. If the verification does not complete within the specified time, the verification fails.

For more information, see “Verification Time Limit” in the *Polyspace Products for Ada Reference*.

Automatic Addition of Specifications for Selected Source Files

When launching a verification from the Eclipse IDE or the Polyspace Verification Environment, the software automatically searches for the package specifications associated with the selected source files, and adds them to the set of sources to verify.

As a result, the verification may contain more source files than you select.

Stubbed Tasks

Programs with stubbed tasks, such as those using Ada rendezvous, previously caused compilation errors. These programs can now be verified.

Scaling Issue for Large Applications with Nested Structures/Arrays

With R2011a, you may experience scaling problems for large applications that manipulate strongly nested structures or arrays. When verifying such applications, the verification may fail during the “Software Safety Analysis Level 0” phase. No verification results are generated, although the data dictionaries (Variable View and Call-Graph View) are accessible.

With previous releases, such applications could be verified, but the verification required several days to complete, and produced results with very low selectivity.

If you experience this problem, MathWorks recommends performing a unit-by-unit verification. For more information, see “Running Verification Unit-by-Unit” in the *Polyspace Products for Ada User’s Guide*.

Compatibility Considerations. Verification may fail for code that was previously verified with an earlier version of the product.

Product Name Change in Files and Folders

The Polyspace product name has changed from “PolySpace” to “Polyspace” in R2011a. This change impacts the name of all files and folders created by the software.

For example:

- `PolySpace-Doc` folder has changed to `Polyspace-Doc`
- `PolySpace_xxxx.log` file has changed to `Polyspace_xxxx.log`

Compatibility Considerations. If you have existing folders that use the previous product name (for example, `PolySpace/PolySpace_Common`) the R2011a installation will continue to use these existing folders. However, any files or folders created during or after installation will use the new name.

If you have any shortcuts or scripts that are case-sensitive, you should update them to use the correct name.

License Manager Support

The License Manager for Polyspace products has been upgraded to FLEXnet 11.9.

You may need to upgrade your FLEXnet server and daemon.

For more information, see "Polyspace License Installation" in the *Polyspace Installation Guide*.

Changes to Verification Results

- "Compatibility Considerations" on page 33
- "Write Access in Data Table with Main Generator and Protected Objects" on page 33
- "NIV for Variables Initialized at Declaration" on page 33
- "Range Error with `greenhills` OS Target" on page 33
- "UOVFL and UNFL Checks Removed" on page 34
- "NIV Checks for Universal Constants" on page 34
- "Constants Defined in Package System" on page 34
- "Initialization of Variables Declared and Assigned in Specs" on page 34
- "Parameterless Protected Procedure as Entry Point" on page 35

Compatibility Considerations. Verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Refer to the following sections for information on the specific changes.

Write Access in Data Table with Main Generator and Protected Objects. In previous releases, when using the `-main-generator` option, an incorrect write access could appear in the data-table when entries of protected objects were called.

In R2011a, the data-table no longer includes locations with critical section names.

There may be less write accesses with protected objects that have defined entries when you use the option `-main-generator`.

NIV for Variables Initialized at Declaration. In previous releases, there is no NIV check when a variable is initialized at the declaration.

In R2011a, verification always puts NIV checks on variables and constants. OVFL checks are not put on constants.

The total number of checks may change when compared with previous releases.

Range Error with `greenhills` OS Target. In previous releases, verification could report a false red when using "`Ada.interrupts.Interrupt_ID`" in source code designed for the greenhills compiler.

In R2011a, when you use the type `Ada.interrupts.Interrupt_ID`, and set the `-OS-target` to `greenhills`, the bounds of the type `Interrupt_ID` have changed.

Verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

UOVFL and UNFL Checks Removed. The software no longer generates UOVFL and UNFL checks, but generates OVFL checks in place of these checks.

The number of checks may decrease compared to previous versions of the software.

NIV Checks for Universal Constants. The software now generates NIV checks for read operations on universal constants. If the constants are used in your code, the NIV checks are green. If the constants are in unreacheable code, the NIV checks are gray.

Verification results may change when compared to previous versions of the software. The number of green and gray checks may be higher compared to the number generated by previous versions of the software.

Constants Defined in Package System. In previous releases, constants defined in package system were ignored.

For example:

```
Max_Binary_Modulus    : constant := 16#100000000#;
Max_Nonbinary_Modulus : constant := 16#FFFFFFFF#;
```

In R2011a, values of constants defined in the package system that are used in the program to be verified are taken into account for the verification. This may impact the results.

Initialization of Variables Declared and Assigned in Specs. In previous releases, verification could incorrectly report a red ZDV check for a variable of a type with default values, when the variable is global.

In R2011a, when using the option `-main-generator`, components with initial values of global variables of composite types that are not initialized are considered full range.

Verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Parameterless Protected Procedure as Entry Point. In previous releases, when a parameterless protected procedure is used as an entry point via the option `-entry-points`, the verification fails.

In R2011a, parameterless protected procedures applied to a global object are now accepted as values of the option `-entry-points`.

Changes to Analysis Options

New Options.

- **Calculate code metrics** (`-code metrics`) – See "Code Metrics" on page 27.
- **Verification time limit** (`-timeout`) – See "Verification Time Limit" on page 30.
- **Number of processes for multiple CPU core systems** (`-max-processes`) – See "Multi-Core Support" on page 28.
- **Less range information** (`-less-range-information`) – See "Enhancements in Run-Time Checks Perspective" on page 29.

Changes to Existing Options. None.

Deprecated Options. None.

Polyspace Server for Ada Product

Code Metrics

Polyspace Metrics now generates Ada code metrics, giving you the number of:

- Protected shared variables
- Unprotected shared variables
- Files
- Lines of code
- Packages

- Packages that appear in `with` statements
- Subprograms that appear in `with` statements

You can view the metrics by:

- Using a Web browser to access the **Code Metrics** view of your project in Polyspace Metrics
- Examining the XML file that the software generates

For more information, see "Setting Up Verification to Generate Metrics" and "Review Code Metrics" in the *Polyspace Products for Ada User's Guide*.

Saving Polyspace Metrics Review

Previously, when you saved your project (**Ctrl+S**) after a review of results from Polyspace Metrics, the software would save your comments and justifications both locally and in the Polyspace Metrics repository.

Now, if you save your project (**Ctrl+S**), the software saves your review to a local folder only. A new button is available on the Run-Time Checks toolbar. If you click this button, the software saves your comments and justifications to a local folder *and* the Polyspace Metrics repository.

This feature allows you to upload your review to the repository only when you are satisfied that your review is, for example, correct and complete.

You can still configure your software to display the previous behavior.

For more information, see "Saving Review Comments and Justifications" in the *Polyspace Products for Ada User's Guide*.

Multi-Core Support

On multi-core computers, you can reduce verification time by specifying the use of core processors simultaneously to perform the verification. The software provides a new command line option –max-processes, which uses a default value of 4. You can specify any value between 1 and 128.

For more information, see “Number of processes for multiple CPU core systems” in the *Polyspace Products for Ada Reference*.

Generated Main with Explicit Tasks and Accept Statements

If you select the option `Generate a main` and there are explicit tasks in the source code, then task bodies are verified like subprograms and `accept` statements are not executed. After verification, code associated with the `accept` statements is gray. For more information, see “Generate a main” in the *Polyspace Products for Ada Reference*.

Compatibility Considerations. Previously, if there were explicit tasks and `accept` statements in the source code, a verification run could have generated red, green, or orange checks for code within the `accept` statements. Now, verification colors the code within these `accept` statements gray.

Automatic Comment Import for Server Verifications

When you download results from the Polyspace server, the software now automatically imports any comments from results in the destination folder into the downloaded results (except for verifications using the option `-add-to-results-repository`).

As a result of this change, you can now download intermediate results for a verification running on the Polyspace server, and add or edit comments on those results. When you later download the final results, your comments are preserved.

You can also download and comment on a single unit of a unit-by-unit verification, even if other units are still pending in the server queue. When you download the final results (which overwrites the earlier results), your comments are preserved.

License Manager Support

The License Manager for Polyspace products has been upgraded to FLEXnet 11.9.

You may need to upgrade your FLEXnet server and daemon.

For more information, see “Polyspace License Installation” in the *Polyspace Installation Guide*.

Version 5.7 (R2011a) Polyspace Model Link Products

This table summarizes what's new in V5.7 (R2011a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Model Link SL Bug Reports Polyspace Model Link TL Bug Reports Polyspace UML Link RH Bug Reports

New features and changes introduced in this version are organized by product:

Polyspace Model Link SL Product

Overflow Check Customization

New options allow you to customize how OVFL checks are handled during verification. You can customize computation through overflow constructions, control the presence of overflow checks, and the dynamic behavior in case of a run-time error.

These options allow you to:

- Not generate OVFL checks on all computations (values are computed the same way processors do).
- Not truncate the value after an OVFL check, and carry on with wrapped values (OVFL check does not impact values during verification).

For more information, see "Detect overflows on (`-scalar-overflows-checks`)" and "Overflows computation mode (`-scalar-overflows-behavior`)" in the *Polyspace Products for C Reference*.

Compatibility Considerations. When using the new options, verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Main Generator Improvements

Enhanced main-generator to improve verification results for generated code.

The new main-generator is specifically designed for cyclic programs, to support generated code and Model-Based Design. The main generator considers the scope of:

- `_step`
- initialization functions
- calibrations

This improves verification results at the subsystem level.

Sample main:

```
initialize_parameters
call_initialization_functions
while (1){
  initialize_inputs
     call_step_functions
 }
call_terminate_functions
```

For more information, see "Main Generation for Model Verification" in the *Polyspace Model Link Products User's Guide*, and "Automatically Generating a Main" in the *Polyspace Products for C User's Guide*.

Compatibility Considerations. Due to precision improvements, verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Data Range Management

Polyspace Model Link™ SL software now allows you to run different modes of verification, such as robustness vs. contextual, by specifying how the verification handles data ranges on model inputs, outputs, and tunable parameters within the model

---

Note The new Data Range Management settings require Simulink Version 7.4 (R2009b) or later.

For more information, see "Data Range Management"in the *Polyspace Model Link Products User's Guide.*

Block Annotation

You can now annotate blocks in your Simulink model to justify known run-time checks or coding-rule violations.

Annotating a block allows you to highlight and categorize checks identified in previous verifications, so that you can focus on new checks when reviewing your verification results.

The Polyspace Run-Time Checks perspective displays the information that you provide, and marks the checks as Justified.

For more information, see "Annotating Blocks to Justify Known Checks or Coding-Rules Violations"in the *Polyspace Model Link Products User's Guide*.

Precision Improvements

Precision enhancements on arrays and functions provide improved Selectivity (less orange) in your verification results.

Compatibility Considerations. Verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Simulink Software Support

Added support for Simulink Version 7.7 (R2011a).

Version 8.0 (R2010b) Polyspace for C/C++ Products

This table summarizes what's new in V8.0 (R2010b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for C/C++ Product" on page 43
- "Polyspace® Server for C/C++ Product" on page 67

Polyspace Client for C/C++ Product

Polyspace Graphical User Interface

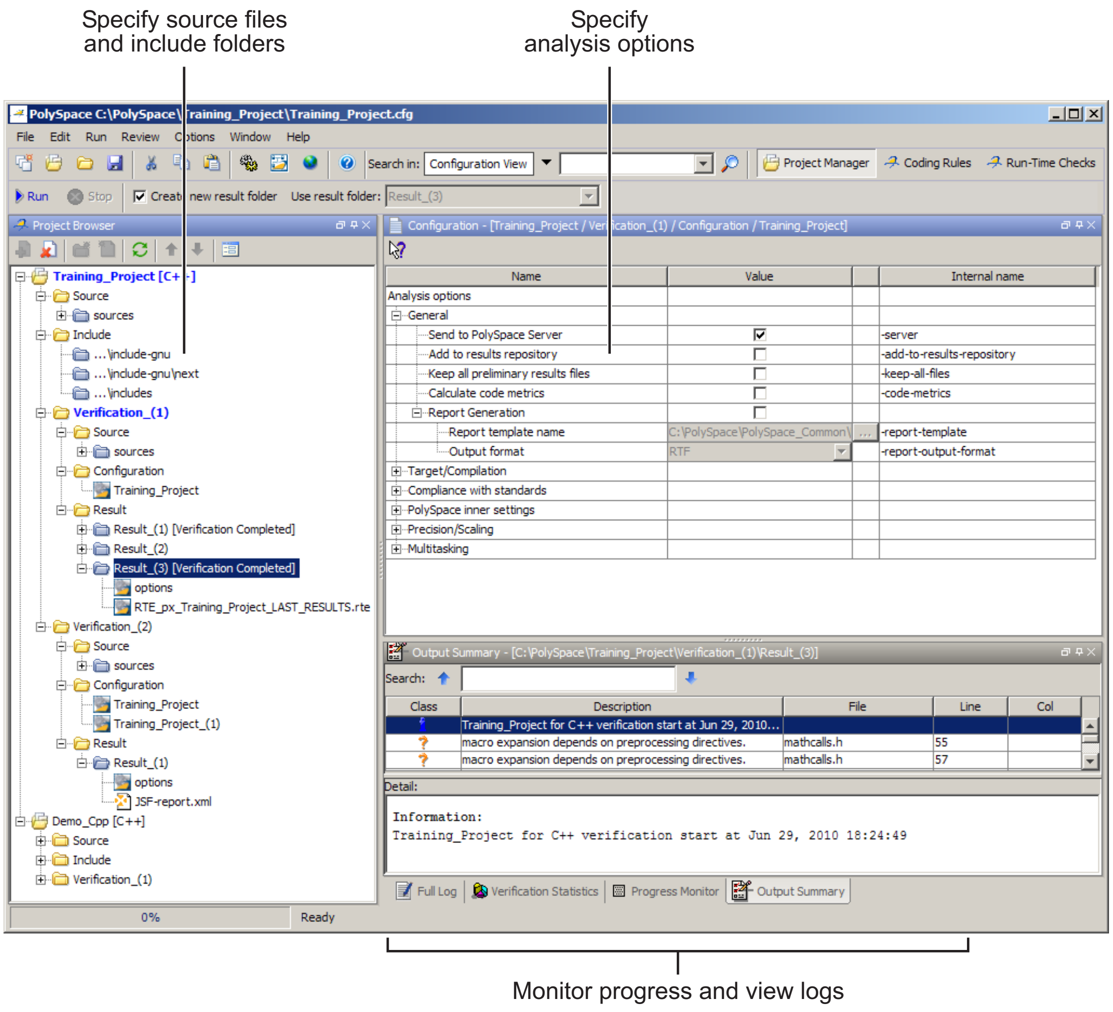
Redesigned Polyspace graphical user interface replaces the Launcher and Viewer modules with a single, unified interface called the Polyspace verification environment (PVE).

You use the Polyspace verification environment to create Polyspace projects, launch verifications, and review verification results. The new interface also enables you to provide comments in the source code or in the results.

The Polyspace verification environment consists of three perspectives:

- "Project Manager Perspective" on page 44
- "Coding Rules Perspective" on page 45
- "Run-Time Checks Perspective" on page 46

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


Specify source files and include folders
Specify analysis options
Monitor progress and view logs
PolySpace C:\PolySpace\Training_Project\Training_Project.cfg
File Edit Run Review Options Window Help
Search in: Configuration View
Project Manager
Coding Rules
Run-Time Checks
Run
Stop
Create new result folder
Use result folder: Result_(3)
Project Browser
Configuration - [Training_Project / Verification_(1) / Configuration / Training_Project]
Output Summary - [C:\PolySpace\Training_Project\Verification_(1)\Result_(3)]
Full Log
Verification Statistics
Progress Monitor
Output Summary
0%
Ready

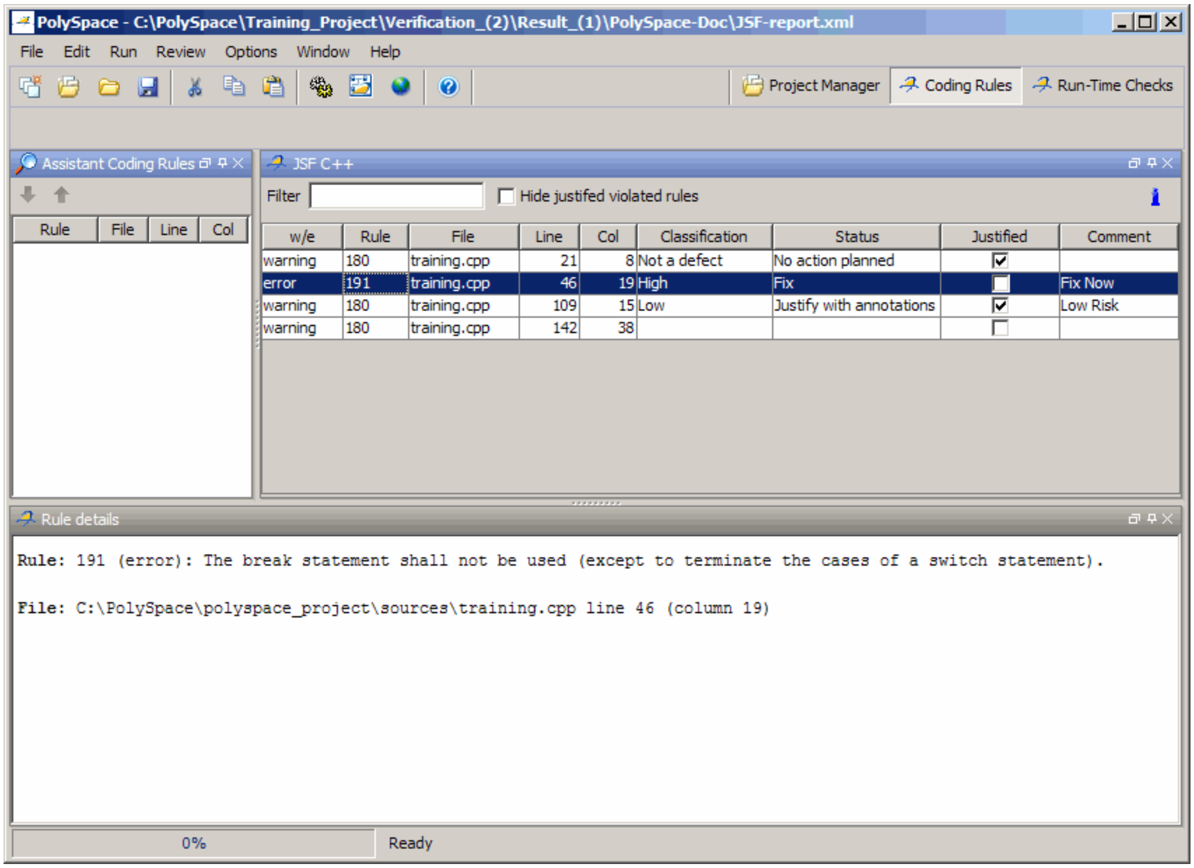
Name	Value	Internal name
Analysis options		
General		
Send to PolySpace Server	✓	-server
Add to results repository	☐	-add-to-results-repository
Keep all preliminary results files	☐	-keep-all-files
Calculate code metrics	☐	-code-metrics
Report Generation	☐	
Report template name	C:\PolySpace\PolySpace_Common\	-report-template
Output format	RTF	-report-output-format
Target/Compilation		
Compliance with standards		
PolySpace inner settings		
Precision/Scaling		
Multitasking		

Class	Description	File	Line	Col
	Training_Project for C++ verification start at Jun 29, 2010...			
?	macro expansion depends on preprocessing directives.	mathcalls.h	55	
?	macro expansion depends on preprocessing directives.	mathcalls.h	57	

```
Information:
Training_Project for C++ verification start at Jun 29, 2010 18:24:49
```

For information on using the Project Manager perspective, see "Setting Up a Verification Project" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Coding Rules Perspective. The Coding Rules perspective allows you to review results from the Polyspace coding rules checker, to ensure compliance with established coding standards.


PolySpace - C:\PolySpace\Training_Project\Verification_(2)\Result_(1)\PolySpace-Doc\JSF-report.xml
File Edit Run Review Options Window Help
Project Manager
Coding Rules
Run-Time Checks
Assistant Coding Rules
Rule File Line Col
JSF C++
Filter
Hide justifed violated rules
Rule details
0%
Ready

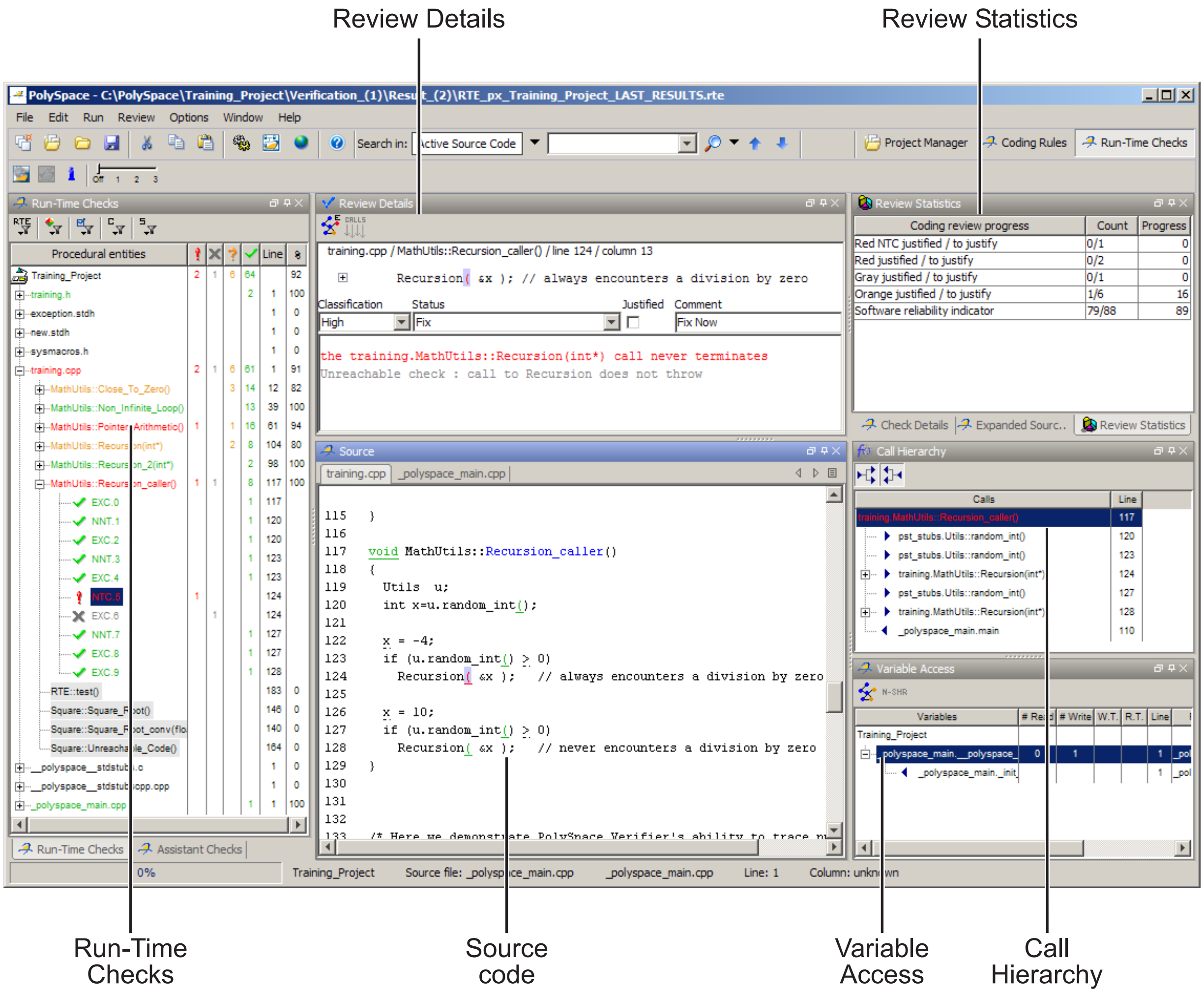
w/e	Rule	File	Line	Col	Classification	Status	Justified	Comment
warning	180	training.cpp	21	8	Not a defect	No action planned	☑	
error	191	training.cpp	46	19	High	Fix	☐	Fix Now
warning	180	training.cpp	109	15	Low	Justify with annotations	☑	Low Risk
warning	180	training.cpp	142	38			☐	

```
Rule: 191 (error): The break statement shall not be used (except to terminate the cases of a switch statement).

File: C:\PolySpace\polyspace_project\sources\training.cpp line 46 (column 19)
```

For information on using the Coding Rules perspective, see “MISRA C Coding Rules Checker” in the *Polyspace Products for C User’s Guide* or “Checking Coding Rules” in the *Polyspace Products for C++ User’s Guide*.

Run-Time Checks Perspective. The Run-Time Checks perspective allows you to review verification results, comment individual checks, and track review progress.



Procedural entities	❗	✖	?	✓	Line	
Training_Project	2	1	6	64		92
training.h				2	1	100
exception.stdh					1	0
new.stdh					1	0
sysmacros.h					1	0
training.cpp	2	1	6	61	1	91
MathUtils::Close_To_Zero()			3	14	12	82
MathUtils::Non_Infinite_Loop()				13	39	100
MathUtils::Pointer_Arithmetic()	1		1	16	61	94
MathUtils::Recursion(int*)			2	8	104	80
MathUtils::Recursion_2(int*)				2	98	100
MathUtils::Recursion_caller()	1	1		8	117	100
EXC.0				1	117	
NNT.1				1	120	
EXC.2				1	120	
NNT.3				1	123	
EXC.4				1	123	
NTC.5	1				124	
EXC.6		1			124	
NNT.7				1	127	
EXC.8				1	127	
EXC.9				1	128	
RTE::test()					183	0
Square::Square_Root()					146	0
Square::Square_Root_conv(flo					140	0
Square::Unreachable_Code()					164	0
__polyspace__stdstubs.c					1	0
__polyspace__stdstubs_cpp.cpp					1	0
_polyspace_main.cpp				1	1	100

0%

training.cpp / MathUtils::Recursion_caller() / line 124 / column 13

```
Recursion( &x ); // always encounters a division by zero
```

Classification	Status	Justified	Comment
High	Fix	☐	Fix Now

```
the training.MathUtils::Recursion(int*) call never terminates
Unreachable check : call to Recursion does not throw
```

Coding review progress	Count	Progress
Red NTC justified / to justify	0/1	0
Red justified / to justify	0/2	0
Gray justified / to justify	0/1	0
Orange justified / to justify	1/6	16
Software reliability indicator	79/88	89

training.cpp | _polyspace_main.cpp

```
115   }
116
117   void MathUtils::Recursion_caller()
118   {
119     Utils  u;
120     int x=u.random_int();
121
122     x = -4;
123     if (u.random_int() > 0)
124       Recursion( &x );   // always encounters a division by zero
125
126     x = 10;
127     if (u.random_int() > 0)
128       Recursion( &x );   // never encounters a division by zero
129   }
130
131
132
133   /* Here we demonstrate PolySpace Verifier's ability to trace n
```

Calls	Line
training.MathUtils::Recursion_caller()	117
pst_stubs.Utils::random_int()	120
pst_stubs.Utils::random_int()	123
training.MathUtils::Recursion(int*)	124
pst_stubs.Utils::random_int()	127
training.MathUtils::Recursion(int*)	128
_polyspace_main.main	110

Variables	# Read	# Write	W.T.	R.T.	Line	
Training_Project						
_polyspace_main.__polyspace_	0	1			1	_pol
_polyspace_main._init_					1	_pol

Training_Project   Source file: _polyspace_main.cpp   _polyspace_main.cpp   Line: 1   Column: unknown

For information on using the Run-Time Checks perspective, see "Reviewing Verification Results" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Permissiveness on File and Folder Names

Polyspace software now allows space characters in the names of Projects, source files, and folders, as well as in option arguments.

In addition, multiple source files with the same name are now allowed.

Note Non-ASCII characters in file names are not supported.

MISRA C++ Coding Rules Support

Enhanced MISRA C++ checker supports all statically enforceable MISRA-C++ coding rules.

Polyspace software can now check all possible C++ programming rules defined by the MISRA® C++ coding standard. The Polyspace MISRA C++ checker provides messages when MISRA C++ rules are not respected. Most messages are reported during the compile phase of a verification.

Note The Polyspace MISRA C++ checker is based on MISRA C++:2008 – "Guidelines for the use of the C++ language in critical systems." For more information on these coding standards, see http://www.misra-cpp.com.

For more information, see "Checking Coding Rules", in the *Polyspace Products for C++ User's Guide*.

Coding Rules Checker Enhancements

The coding rules checker for MISRA C, MISRA C++, and JSF C++ coding standards has been enhanced as follows:

- You can now set all supported coding rules to any state: `Error`, `Warning`, or `Off`.
- The **Files and Folders to ignore** (`-includes-to-ignore`) option now supports the keyword "`all`," allowing you to exclude all included files from coding rules checking.

- The new Coding Rules perspective allows you to review and categorize coding rule violations, and provide comments in the results to justify violations.
- The MISRA C checker now allows you to automatically select two recommended subsets of coding rules (`SQO-subset1`, and `SQO-subset2`), in addition to creating a custom subset.

For more information, see "MISRA C Coding Rules Checker" in the *Polyspace Products for C User's Guide* or "Checking Coding Rules", in the *Polyspace Products for C++ User's Guide*.

Code Metrics (for C)

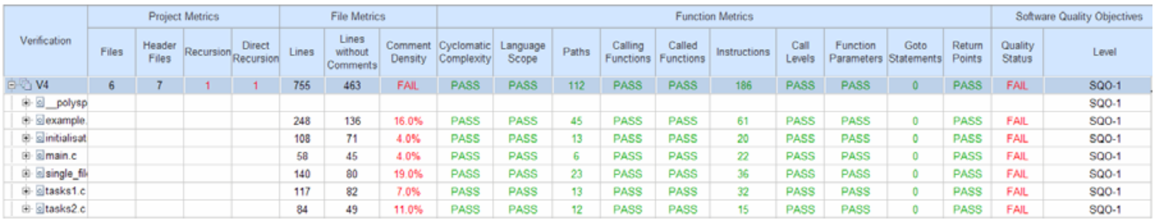
Code metric support, including cyclomatic number and other HIS metrics.

Polyspace verification can now generate metrics about code complexity, which are based on the Hersteller Initiative Software (HIS) standard.

These metrics include:

- **Project metrics** – including number of recursions, number of include headers, and number of files.
- **File metrics** – including comment density, and number of lines.
- **Function metrics** – including cyclomatic number, number of static paths, number of calls, and Language scope.

When you run a verification with the `-calculate-code-metrics` option enabled, you can view software quality metrics data in the Polyspace Metrics Web interface (**Code Metrics** view), or by running a Software Quality Objectives report from the Polyspace verification environment.



Verification	Project Metrics				File Metrics			Function Metrics											Software Quality Objectives	
	Files	Header Files	Recursion	Direct Recursion	Lines	Lines without Comments	Comment Density	Cyclomatic Complexity	Language Scope	Paths	Calling Functions	Called Functions	Instructions	Call Levels	Function Parameters	Goto Statements	Return Points	Quality Status	Level
V4	6	7	1	1	755	463	FAIL	PASS	PASS	112	PASS	PASS	186	PASS	PASS	0	PASS	FAIL	SQO-1
__polysp																			SQO-1
example.					248	136	16.0%	PASS	PASS	45	PASS	PASS	61	PASS	PASS	0	PASS	FAIL	SQO-1
initialisat					108	71	4.0%	PASS	PASS	13	PASS	PASS	20	PASS	PASS	0	PASS	FAIL	SQO-1
main.c					58	45	4.0%	PASS	PASS	6	PASS	PASS	22	PASS	PASS	0	PASS	FAIL	SQO-1
single_fil					140	80	19.0%	PASS	PASS	23	PASS	PASS	36	PASS	PASS	0	PASS	FAIL	SQO-1
tasks1.c					117	82	7.0%	PASS	PASS	13	PASS	PASS	32	PASS	PASS	0	PASS	FAIL	SQO-1
tasks2.c					84	49	11.0%	PASS	PASS	12	PASS	PASS	15	PASS	PASS	0	PASS	FAIL	SQO-1

The software generates numeric values or pass/fail results for various metrics.

For more information, see “Software Quality with Polyspace Metrics”in the *Polyspace Products for C User’s Guide* or *Polyspace Products for C++ User’s Guide*.

Filtering Orange Checks Caused by Input Data (New for C++)

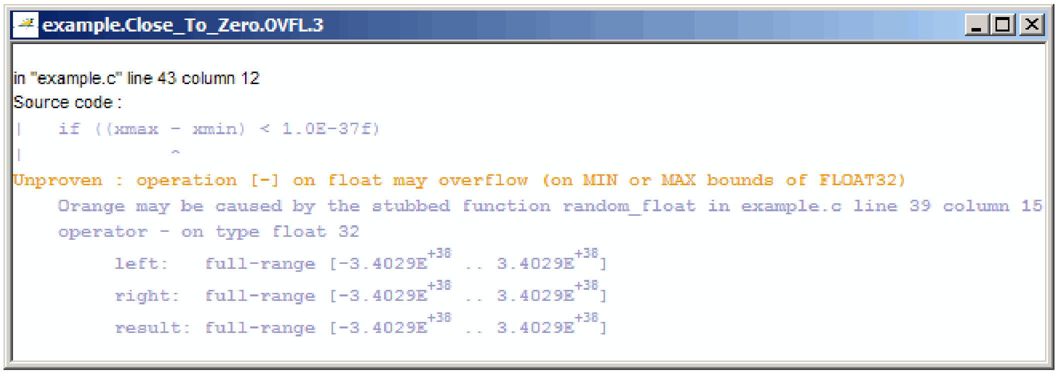
Polyspace verification now identifies orange checks caused by input data for C++ code, in addition to C code. The software provides additional information on these orange checks, and allows you to hide them in the Run-Time Checks perspective.

Note Although this type of orange check could reveal a bug, they usually do not.

Verification can identify orange checks caused by:

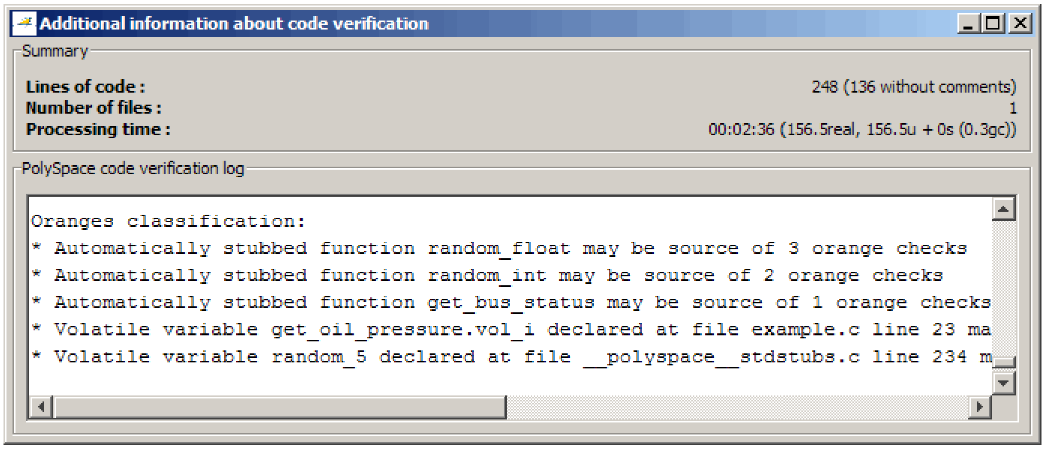
- Stubs
- Main-generator calls
- Volatile variables
- Extern variables
- Absolute address

When the software identifies this type of orange check, the Run-Time Checks perspective provides information on its cause.



```
in "example.c" line 43 column 12
Source code :
|    if ((xmax - xmin) < 1.0E-37f)
|             ^
Unproven : operation [-] on float may overflow (on MIN or MAX bounds of FLOAT32)
    Orange may be caused by the stubbed function random_float in example.c line 39 column 15
    operator - on type float 32
         left:   full-range [-3.4029E+38 .. 3.4029E+38]
         right:  full-range [-3.4029E+38 .. 3.4029E+38]
         result: full-range [-3.4029E+38 .. 3.4029E+38]
```

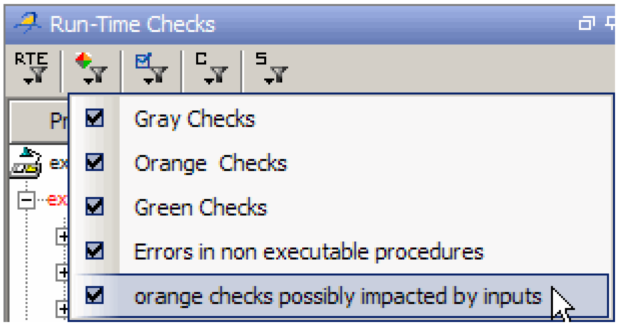
The Polyspace code verification log file also lists possible sources of imprecision for orange checks.


Additional information about code verification
Summary
PolySpace code verification log

Lines of code :	248 (136 without comments)
Number of files :	1
Processing time :	00:02:36 (156.5real, 156.5u + 0s (0.3gc))

```
Oranges classification:
* Automatically stubbed function random_float may be source of 3 orange checks
* Automatically stubbed function random_int may be source of 2 orange checks
* Automatically stubbed function get_bus_status may be source of 1 orange checks
* Volatile variable get_oil_pressure.vol_i declared at file example.c line 23 ma
* Volatile variable random_5 declared at file __polyspace__stdstubs.c line 234 m
```

In addition, you can now hide these types of orange checks in the Run-Time Checks perspective. When using Expert mode, click the **Color filter** icon, then clear the **Orange checks possibly impacted by inputs** option.

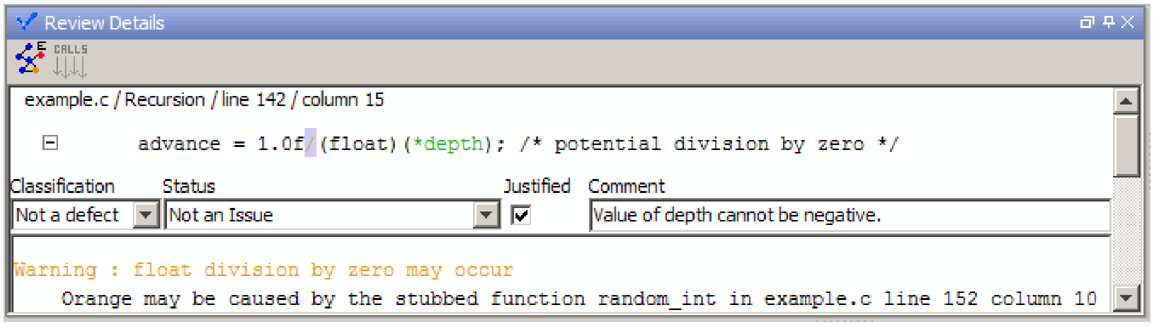

Run-Time Checks
RTE
C
S
Gray Checks
Orange Checks
Green Checks
Errors in non executable procedures
orange checks possibly impacted by inputs

The software hides orange checks impacted by inputs.

For more information, see "Working with Orange Checks Caused by Input Data" in the *Polyspace Products for C++ User's Guide*.

New Options to Classify Run-Time Checks and Coding Rules Violations

When reviewing results in the Run-Time Checks perspective or the Coding Rules perspective, the software now provides additional options for classifying checks



example.c / Recursion / line 142 / column 15

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

Classification	Status	Justified	Comment
Not a defect	Not an Issue	✓	Value of depth cannot be negative.

```
Warning : float division by zero may occur
   Orange may be caused by the stubbed function random_int in example.c line 152 column 10
```

After you review the check, you can specify the following:

- **Classification** – Select an option to describe the seriousness of the issue.
- **Status** – Select an option to describe how you intend to address the issue.
- **Justified** – Select the check box to indicate that you have justified this check or rule violation.

- **Comment** – Enter additional information about the check

The software provides predefined values for Classification and Status. You can also define your own statuses.

In addition to reviewing checks through the user interface, you can place comments in your code that highlight and categorize checks identified in previous verifications. The software displays the information that you provide within your code comments, and marks the checks as **Justified**.

For more information, see “Reviewing and Commenting Checks ” in the *Polyspace Products for C User’s Guide* or *Polyspace Products for C++ User’s Guide*.

Compatibility Considerations. The syntax for code comments has changed to reflect the new options for categorizing checks.

The syntax for run-time checks is now:

```
/* polyspace<RTE:RTE1 : [Classification] :
[Status] > [Comment] */
```

The syntax for coding-rule violations is now:

```
/* polyspace<JSF:Rule1 : [Classification] :
[Status] > [Comment] */
```

If you placed comments in your code using the previous syntax, the comments will still appear in your results, but the text may be displayed in different columns.

For more information on code comments, including full syntax, see “Highlighting Known Coding Rule Violations and Run-Time Errors” in the *Polyspace Products for C User’s Guide* or *Polyspace Products for C++ User’s Guide*.

Japanese and Korean Text in Comments

Japanese and Korean characters are now supported for comments in results review.

For more information, see “Reviewing Checks”in the *Polyspace Products for C User’s Guide* or *Polyspace Products for C++ User’s Guide*.

Pointer Information in the Run-Time Checks Perspective

Enhanced ToolTip messages on pointers to improve understanding of problems with the pointer.

For example, messages on offset in the allocated buffer now indicate if the pointer is inside its bounds, in addition to giving raw numbers.

For more information, see “Using Pointer Information in Run-Time Checks Perspective” in the *Polyspace Products for C User’s Guide* or *Polyspace Products for C++ User’s Guide*.

Main Generation in C++

Enhanced main generation options in C++ allow you to use both main generator and class analyzer modes at the same time (the options `-class-analyzer` and `-main-generator-calls` can be used simultaneously).

In addition, the options **Select methods called by the generated main** (`-class-analyzer-calls`), and **Function calls** (`-main-generator-calls`) are enhanced to provide more flexibility in configuring what functions are called. by the generated main.

The default behavior of the main generator is now as follows:

- If you set the **Class name** (`-class-analyzer`) option to `all` or `custom`, and set `-class-analyzer-calls`, then the option `-main-generator-calls` is automatically set to `unused`, unless you explicitly set another value for `-main-generator-calls`.
- Setting the **Function calls** (`-main-generator-calls`) option to `unused`, `all`, or `custom` automatically sets `-class-analyzer` to `none`, unless you explicitly set the `-class-analyzer` option.

For more information, see “Generate a main (`-main-generator`)”in the *Polyspace Products for C++ Reference*.

Compatibility Considerations. If you use scripts that specify a value for the option `-class-analyzer-calls`, you may need to update your scripts to reflect the new option arguments. The new syntax is:

```
-class-analyzer-calls [ all | unused | inherited_all |
  inherited_unused | custom ]
```

Where:

- `all` corresponds to the previous argument "`default`."
- `inherited_unused` corresponds to previous argument "`inherited`."
- `inherited_all` means every inherited methods will be called by the generated main.

Multiple Functions Called Before Main

The option **First functions to call** (`-function-called before main`) now accepts a list of multiple functions, instead of just a single function.

For more information, see“Functions called after loop (`-function-called-after-loop`)” in the *Polyspace Products for C Reference* or “Generate a main (`-main-generator`)”in the *Polyspace Products for C++ Reference.*

Support for C99 Extensions (C)

Partial support of C99 extensions.

A new option, `-allow-language-extensions`, enables verification to accept a subset of common C language constructs and extended keywords, as defined by the C99 standard or supported by many compilers.

When you select this option, the following constructs are supported:

- Designated initializers (labeling initialized elements)
- Compound literals (structs or arrays as values)
- Boolean type (`_Bool`)
- Statement expressions (statements and declarations inside expressions)

- `typeof` constructs
- Case ranges
- Empty structures
- Cast to union
- Local labels (`__label__`)
- Hexadecimal floating-point constants
- Extended keywords, operators, and identifiers (`_Pragma`, `__func__`, `__const__`, `__asm__`)

In addition, when you use this option, the software ignores the following extended keywords:

- `near`
- `far`
- `restrict`
- `_attribute_(X)`
- `rom`

For more information, see "Allow language extensions (`-allow-language-extensions`)"in the *Polyspace Products for C Reference*.

New Target Processor Support (C)

Added support for 64-bit target.

The Target processor type (`-target`) option now supports the target x86_64, allowing the verification to emulate 64–bit processors.

For more information, see "Predefined Target Processor Specifications" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Default Target Processor

The default setting of the Target processor type (`-target-processor`) option has changed from `SPARC` to `i386`.

Compatibility Considerations. If you launch verifications without specifying a value for this option, the default value has changed. Therefore, your verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Default Operating System Target

The default setting of the Operating system target for Polyspace stubs (`-OS-target`) option has changed from `Solaris` to `Linux`.

Compatibility Considerations. If you launch verifications without specifying a value for this option, the default value has changed. Therefore, your verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Include Folders Added to Verification by Default

Polyspace software now automatically adds the following standard include folders after any includes you specify:

- *`PolySpace_Install`*`/Verifier/include/include-gnu`
- *`PolySpace_Install`*`/Verifier/include/include-gnu/next`

The path to these folders will be printed in the log file at the beginning of the compilation.

Compatibility Considerations. The total number of checks in your verification may change when compared to previous releases, if you did not previously include these folders.

Operating System Support

Added support for the Windows® 7 operating system.

Solaris™ operating system is no longer supported for new installations.

For more information, see the *Polyspace Installation Guide*.

Changes to Verification Results

Compatibility Considerations. Verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Refer to the following sections for information on the specific changes.

New NIP Check on Pointer to Member Function. New NIP check introduced on variables corresponding to a pointer to member function when verifying the pointer.

Previously, pointers to member functions were translated into a structure composed of 4 fields. In this release, these 4 fields are checked and the information is merged into a NIP.

For example (in R2010a and earlier):

```
struct A {
  virtual void f() { } ;
  void g() {} ;
};

int main()
{
    A a ;
    void (A::*pmf)() ;
    volatile int alea ;

    if (alea)
      assert(pmf != 0) ; // RED NIV located on '(' => expected a NIP

    if (alea)
      (a.*pmf)();   // no red, only grey

    if (alea)
      pmf = &A::f ;
    else
      pmf = &A::g ;

    assert(pmf != 0) ; // spurious info on '(', green NIV instead of NIP

    (a.*pmf)();  //


  }
```

In R2010b, some NIV checks may change to NIP checks (on pointer to member function) The Selectivity rate of your results may change when compared to previous versions of the software.

Generated Main Calls in the Main Loop and init Function. The call of a function given to the option `-function-called-before-main` is now removed from the main-generator loop.

In previous releases, when an "init" function was called before the main loop, it was also called in the main loop. For example:

```
void main(void)
{
    /* ********************************************** *
    * Initialization of global variables with random *
    * ********************************************** */

    /* ****************************** *
```

```
        * Call of initialization function *
        * ******************************** */
        {

            /* call it */
            init();
        }
        while (PST_TRUE())
        {
            /* **************** *
            * Call of functions *
            * **************** */
            if (PST_TRUE())
            {

                /* call it */
                init();
            }
            if (PST_TRUE())
            {

                /* call it */
                foo();
            }
        }
    }
```

This init function is now removed from the main-generator loop.

The Selectivity rate of your results may change when compared to previous versions of the software.

INF Checks Replaced by Value on Range (C++). When transforming C checks into C++ checks, the software changes INF checks into a new value on range category (VOBJ) and displays them like other value on range (VOR) information in the Run-Time Checks perspective.

The number of checks in your results may decrease when compared to previous releases.

Value on Range (VOR) Values in pass0 Results. Verification results now give value on range values (intervals) computed during pass0.

In previous releases, value on range values in pass0 could only be constants or the type full-range.

When reviewing pass0 results, value on range tooltips will now contain more information than in previous releases.

Changes in Behavior of Inline and Sensitivity Context Options. Verification now displays a warning of you specify a nonexistent function as an argument of the options **Inline** (-inline) or **Sensitivity context** (-context-sensitivity). The option is ignored, and verification continues.

In previous releases, specifying a nonexistent function caused the verification to stop.

Permissiveness on Delete of Pointer to Incomplete Class. Polyspace verification now gives a warning when it detects a delete on a pointer with incomplete class, unless you set the Dialect (-dialect) option to iso. If you specify the iso dialect, the verification will raise a compilation error.

In previous releases, a delete on a pointer with incomplete class implied a crash, and produced an error. For example:

```
#include <memory>

typedef class BaseClass;
typedef class Container
{
private:
    std::auto_ptr<BaseClass> data;

public:
    Container(std::auto_ptr<BaseClass> p) : data(p) {};
};
```

In R2010b, this code will be accepted with a warning, except in iso mode, where it will raise a compilation error.

Changes to Coding Rules Checker Results

- "Compatibility Considerations" on page 61
- "MISRA and JSF Violations No Longer Reported on Internal Include Folders" on page 61
- "MISRA-C++ Rule 2-10-2 Violations on Type Hidden by Using Directive" on page 61
- "MISRA-C++ Rules 2-10-4 and 2-10-6 Violations on Templates" on page 62
- "MISRA-C++ Rule 3-1-1 Duplicate Violations" on page 63
- "MISRA-C++ Rule 3-4-1 Violations on Local Variables" on page 63
- "MISRA-C++ Rule 7-4-3 Violations on Assembly Language" on page 64
- "MISRA-C++ Rule 12-1-1, 12-1-2, and 12-8-2 Violations " on page 64
- "JSF Rule AV-136 Violations on Local Variables" on page 66

Compatibility Considerations. Due to changes in the coding rules checker, the number of coding rule violations may change when compared to previous versions of the software.

Refer to the following sections for information on the specific changes.

MISRA and JSF Violations No Longer Reported on Internal Include Folders. The coding rules checker now ignores the Include folders provided with the product (`include-gnu/` and `include-linux/`).

No violations are reported for identifiers appearing in hidden files, even if these files are hidden in a hard-coded way.

The total number of violations reported by the coding rules checker may decrease when compared to previous releases, since any violations within the include files are no longer reported.

MISRA-C++ Rule 2-10-2 Violations on Type Hidden by Using Directive. The MISRA-C++ checker is more precise on violations of rule 2-10-2, "Identifiers declared in an inner scope shall not hide an identifier declared in an outer scope," when the type is hidden by a using directive on the same type.

For example:

```
#include "misra.h"

namespace ns1 {
  class A    // No Violation since the type A is declared only here.
  {
    A & operator= ( A const & rhs );
  public:
    A ( );
    virtual void bar( ) = 0;

  };

}
using ns1::A;
namespace ns2 {
  class D : public A
  {
  public:
    virtual void foo( ) = 0;
    D ( ) : A()
    {
    }
  };
}
```

In previous releases, the MISRA-C++ checker incorrectly reported a violation on the type A.

You may see fewer violations of rule 2-10-2 in MISRA C++ reports, when compared with previous releases.

MISRA-C++ Rules 2-10-4 and 2-10-6 Violations on Templates. The coding rules checker no longer reports violations of MISRA-C++ Rules 2-10-4 "A class, union or enum name (including qualification, if any) shall be a unique identifier," and 2-10-6 "If an identifier refers to a type, it shall not also refer to an object or a function in the same scope" when the template class is present in the code. A violation is reported only for explicit specialization (which has its own declaration).

You may see fewer violations of rules of 2-10-4 and 2-10-6 in MISRA C++ reports, when compared with previous releases.

MISRA-C++ Rule 3-1-1 Duplicate Violations. The coding rules checker no longer reports duplicate violations of MISRA-C++ Rule 3-1-1 ”It shall be possible to include any header file in multiple translation units without violating the One Definition Rule.”

In previous releases, the coding rules checker sometimes incorrectly reported this violation multiple times on the same function.

You may see fewer violations of rule 3-1-1 in MISRA C++ reports, when compared with previous releases.

MISRA-C++ Rule 3-4-1 Violations on Local Variables. The MISRA-C++ coding rules checker is more precise on violations of Rule 3-4-1, “An identifier declared to be an object or type shall be defined in a block that minimizes its visibility.”

For example, the coding rules checker no longer reports violations of rule 3-4-1 for the following code:

```
volatile int32_t rd;
if (rd != 0) {
  int32_t i;
  {
    int32_t j;
    {
      goto L1;
    }
    rd = j;
  }
  rd = i;
}
```

In previous releases, the coding rules checker incorrectly reported a violation of rule 3-4-1 for the variable `rd`.

You may see fewer violations of rule 3-4-1 in MISRA C++ reports.

MISRA-C++ Rule 7-4-3 Violations on Assembly Language. The MISRA-C++ checker no longer reports errors for rule 7-4-3, “Assembly language shall be encapsulated and isolated,” for certain compliant constructions. For example:

```
void Delay_a ( void )
{
   asm ( "NOP" );   // Compliant
}
```

In previous releases, the MISRA-C++ checker incorrectly reported a violation of rule 7-4-3 for this code.

You may see fewer violations of rule 7-4-3 in MISRA C++ reports.

MISRA-C++ Rule 12-1-1, 12-1-2, and 12-8-2 Violations . The MISRA-C++ checker is more precise on violations of rule 12-1-1, “An object’s dynamic type shall not be used from the body of its constructor or destructor,” rule 12-1-2 “All constructors of a class should explicitly call a constructor for all of its immediate base classes and all virtual base classes,” and rule 12-8-2 “The copy assignment operator shall be declared protected or private in an abstract class.”

Violations of rule 12-1-1 are now reported on destructors. For example:

```
class C2
  {
  public:
    ~C2 ( )
    {
      typeid ( C2 ); // New 12-1-1 violation reported here
      C2::foo ( );
      foo ( );
      dynamic_cast< C2* > ( this );
    }
    virtual void foo ( );
    C2 ( )
    {
      typeid ( C2 ); // 12-1-1 violation reported
      C2::foo ( );
      foo ( );
      dynamic_cast< C2* > ( this );
    }
  };
```

In addition, violations of these rules are now reported in the following cases:

- On `typeid` on any class with virtual function in itself or in one of its base.
- On `typeid` on pointer this or conversion of pointer this.
- On `dynamic_cast` on pointer this or conversion of pointer this.

For example, in the following code violations are now reported on typeid if the type is struct:

```
struct S2
  {
    ~S2 ( )
    {
      typeid ( S2 ); // New violation reported here
      S2::foo ( );
      foo ( );
      dynamic_cast< S2* > ( this );
    }
    virtual void foo ( );
    S2 ( )
    {
      typeid ( S2 ); // New violation reported here
      S2::foo ( );
      foo ( );
      dynamic_cast< S2* > ( this );
    }
  };
```

In previous releases, the MISRA-C++ checker did not report these violations.

You may see additional violations of rule 12-1-1, 12-1-2, and 12-8-2 in MISRA C++ reports, when compared with previous releases.

JSF Rule AV-136 Violations on Local Variables. The JSF C++ coding rules checker is more precise on violations of Rule 136, "Declarations should be at the smallest feasible scope."

For example, the coding rules checker no longer reports violations of rule 136 for the following code:

```
volatile int32_t rd;
if (rd != 0) {
  int32_t i;
  {
    int32_t j;
    {
      goto L1;
    }
    rd = j;
  }
  rd = i;
}
```

In previous releases, the coding rules checker incorrectly reported a violation of rule 136 for the variable `rd`.

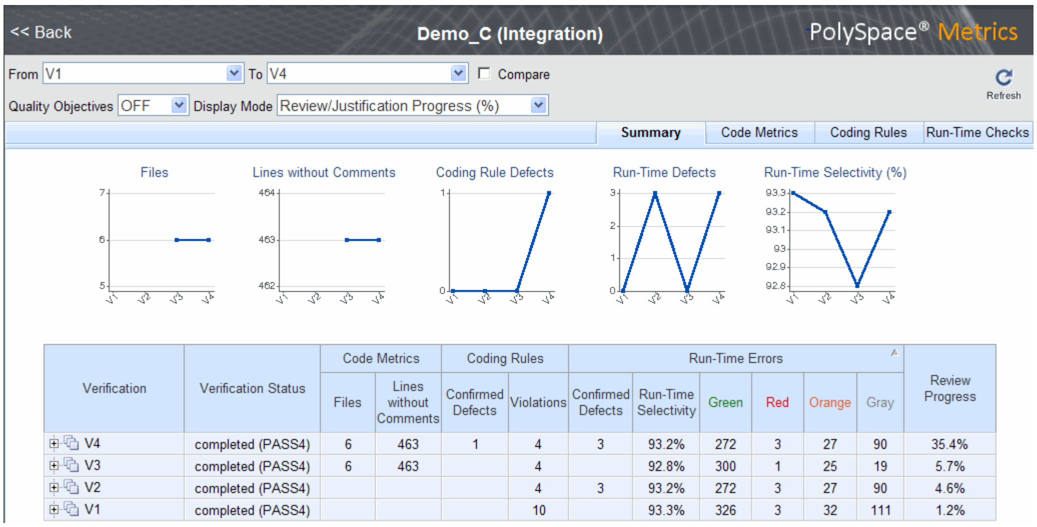
You may see fewer violations of rule 136 in JSF C++ reports.

Polyspace Server for C/C++ Product

Polyspace Metrics Web Interface

A web-based tool for software development managers, quality assurance engineers, and software developers, which allows you to do the following in software projects:

- Evaluate software quality metrics
- Monitor the variation of code metrics, coding rule violations, and run-time checks through the lifecycle of a project
- View defect numbers, run-time reliability of the software, review progress, and the status of the code with respect to software quality objectives.



From V1 To V4 ☐ Compare

Quality Objectives OFF Display Mode Review/Justification Progress (%)

Verification	Verification Status	Code Metrics		Coding Rules		Run-Time Errors						Review Progress
		Files	Lines without Comments	Confirmed Defects	Violations	Confirmed Defects	Run-Time Selectivity	Green	Red	Orange	Gray	
V4	completed (PASS4)	6	463	1	4	3	93.2%	272	3	27	90	35.4%
V3	completed (PASS4)	6	463		4		92.8%	300	1	25	19	5.7%
V2	completed (PASS4)				4	3	93.2%	272	3	27	90	4.6%
V1	completed (PASS4)				10		93.3%	326	3	32	111	1.2%

In addition, if you have the Polyspace® Client™ for C/C++ product installed on your computer, you can drill down to coding rule violations and run-time checks in the Polyspace verification environment. This allows you to:

- Review coding rule violations
- Review run-time checks and, if required, classify these checks as defects

For more information, see Software Quality with Polyspace Metrics in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Automatic Verification

Configure verifications to start automatically and periodically, for example, at a specific time every night. At the end of each verification, the software stores results in a results repository and updates the metrics for your software project. You can also configure the software to send you an email at the end of the verification. This email contains links to results, compilation errors, run-time errors, or processing errors.

For more information, see Specifying Automatic Verification in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Operating System Support

Added support for the Windows 7 operating system.

Solaris operating system is no longer supported for new installations.

For more information, see the *Polyspace Installation Guide*.

Version 6.0 (R2010b) Polyspace for Ada Products

This table summarizes what's new in V6.0 (R2010b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for Ada Product" on page 69
- "Polyspace® Server for Ada Product" on page 75

Polyspace Client for Ada Product

Polyspace Graphical User Interface

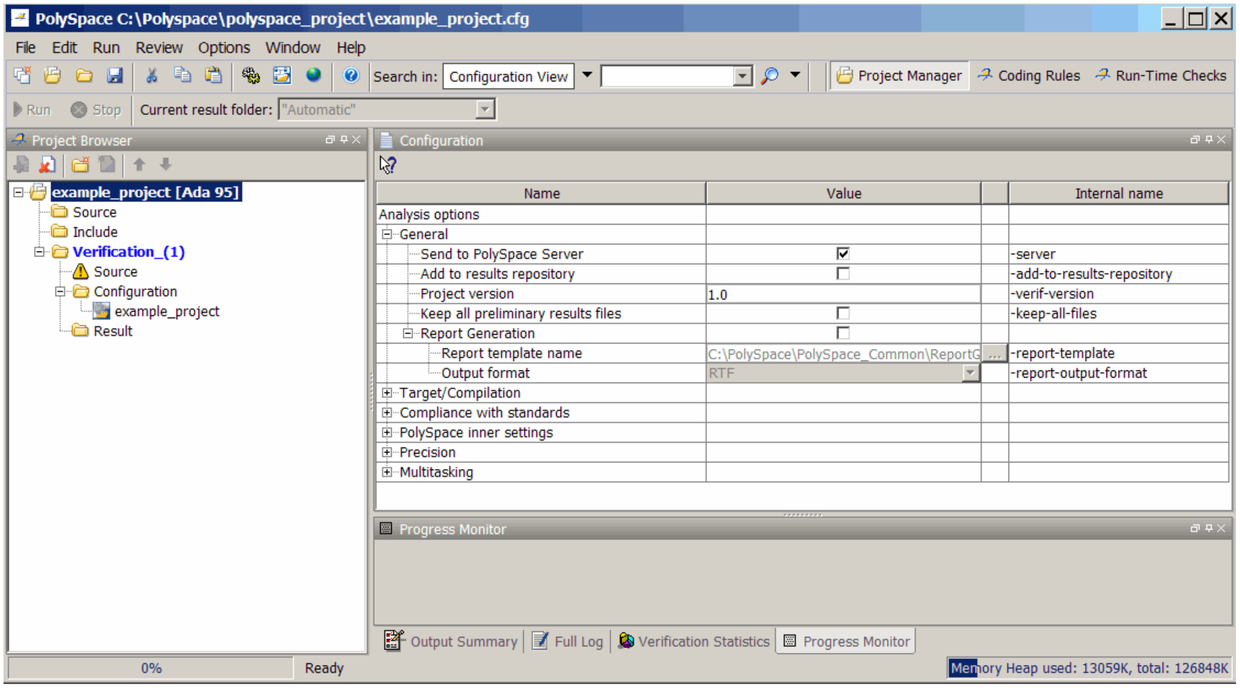
Redesigned Polyspace graphical user interface replaces the Launcher and Viewer modules with a single, unified interface called the Polyspace verification environment (PVE).

You use the Polyspace verification environment to create Polyspace projects, launch verifications, and review verification results. The new interface also enables you to provide comments in the source code or in the results.

The Polyspace verification environment consists of two perspectives:

- "Project Manager Perspective" on page 70
- "Run-Time Checks Perspective" on page 71

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

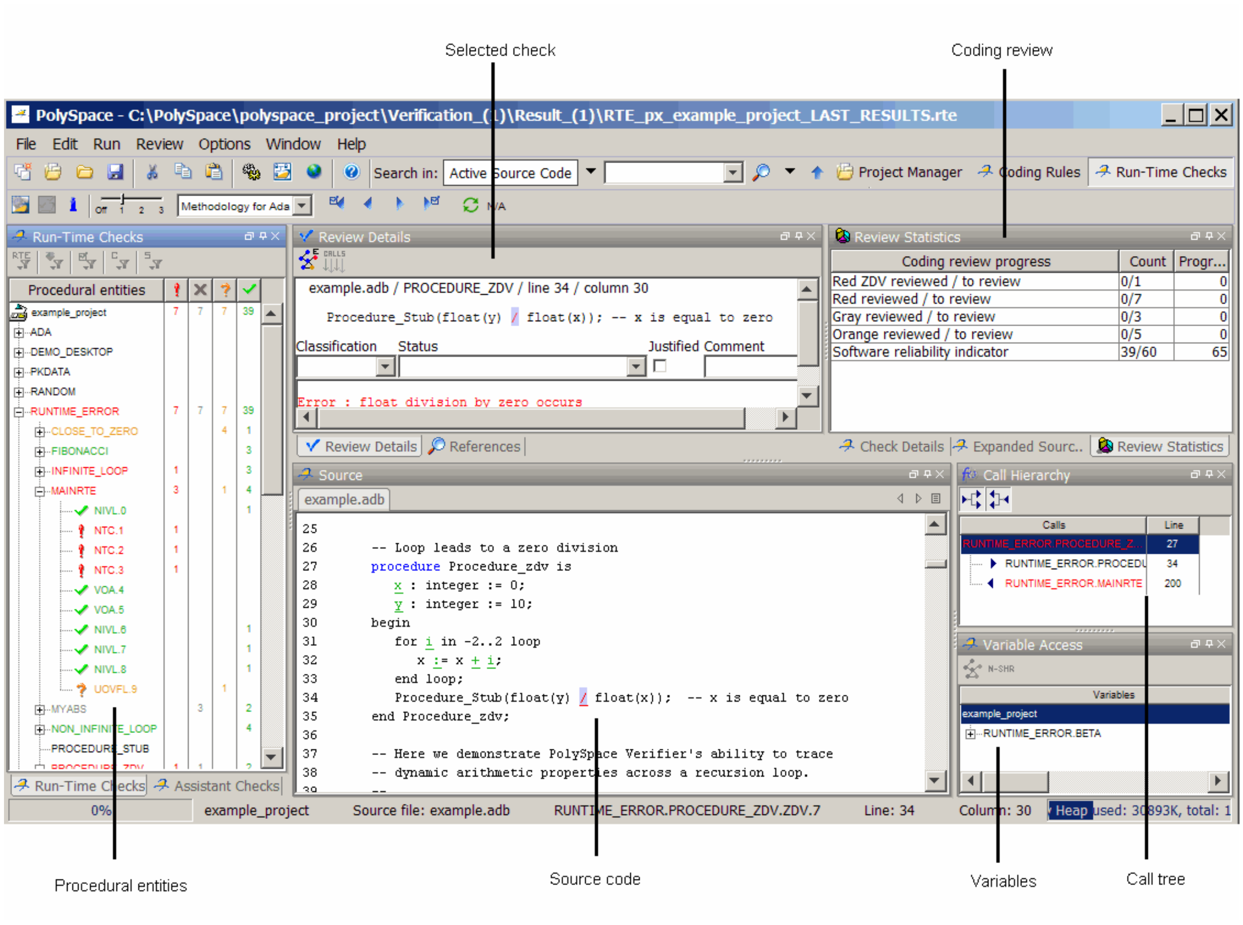


```
example_project [Ada 95]
    Source
    Include
    Verification_(1)
        Source
        Configuration
            example_project
        Result
```

Name	Value		Internal name
Analysis options			
General			
Send to PolySpace Server	☑		-server
Add to results repository	☐		-add-to-results-repository
Project version	1.0		-verif-version
Keep all preliminary results files	☐		-keep-all-files
Report Generation	☐		
Report template name	C:\PolySpace\PolySpace_Common\ReportG	...	-report-template
Output format	RTF		-report-output-format
Target/Compilation			
Compliance with standards			
PolySpace inner settings			
Precision			
Multitasking			

For information on using the Project Manager perspective, see “Setting Up a Verification Project” in the *Polyspace Products for Ada User’s Guide*.

Run-Time Checks Perspective. The Run-Time Checks perspective allows you to review verification results, comment individual checks, and track review progress.



Review Details:

example.adb / PROCEDURE_ZDV / line 34 / column 30

```
Procedure_Stub(float(y) / float(x)); -- x is equal to zero
```

Classification Status Justified Comment

```
Error : float division by zero occurs
```

Review Statistics:

Coding review progress	Count	Progr...
Red ZDV reviewed / to review	0/1	0
Red reviewed / to review	0/7	0
Gray reviewed / to review	0/3	0
Orange reviewed / to review	0/5	0
Software reliability indicator	39/60	65

Source (example.adb):

```
    -- Loop leads to a zero division
    procedure Procedure_zdv is
       x : integer := 0;
       y : integer := 10;
    begin
       for i in -2..2 loop
          x := x + i;
       end loop;
       Procedure_Stub(float(y) / float(x));  -- x is equal to zero
    end Procedure_zdv;

    -- Here we demonstrate PolySpace Verifier's ability to trace
    -- dynamic arithmetic properties across a recursion loop.
    --
```

Call Hierarchy:

Calls	Line
RUNTIME_ERROR.PROCEDURE_Z..	27
RUNTIME_ERROR.PROCEDU	34
RUNTIME_ERROR.MAINRTE	200

Variable Access: example_project, RUNTIME_ERROR.BETA

Status bar: 0% example_project Source file: example.adb RUNTIME_ERROR.PROCEDURE_ZDV.ZDV.7 Line: 34 Column: 30 Heap used: 30893K, total: 1

For information on using the Run-Time Checks perspective, see “Reviewing Verification Results” in the *Polyspace Products for Ada User’s Guide*.

Data Range Specifications

The Data Range Specifications (DRS) feature allows you to set constraints on data ranges using a text file. With this text file, you can specify data ranges for global variables, stubbed functions and procedures, and function call inputs.

For more information, see Specifying Data Ranges for Variables, Functions, and Procedures in the *Polyspace Products for Ada User's Guide*.

Extended Support for Tasks

You can now verify Ada code that contains the following task-related features :

- Pointers to explicit tasks
- Private task types
- Tasks with discriminants
- Entry families
- Explicit tasks declared in a package – you can now verify the package with the `-main-generator` option

Previously, these features would generate errors during a verification.

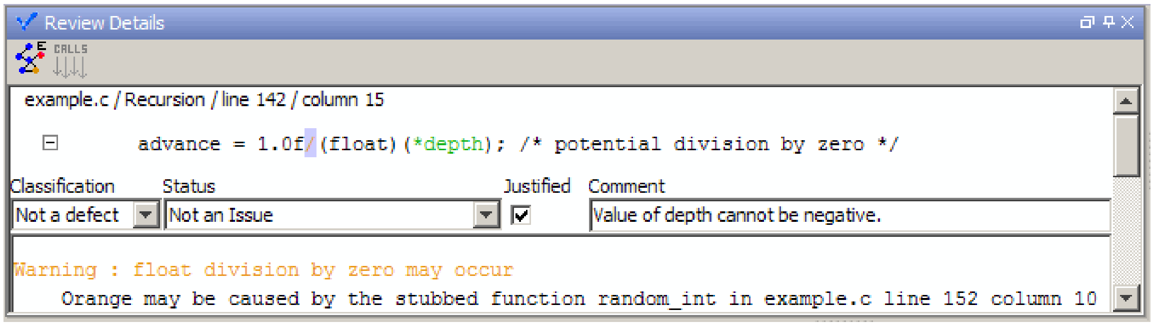
Preprocessor Macros for Compilation

You can now specify, with the `-D` option, the use of preprocessor macros in code compilation. The `-U` option, which nullifies this `-D` option, is also available.

For more information, see "Defined Preprocessor Macros"(-D compiler-flag) and "Undefined Preprocessor Macros" (-U compiler-flag) in the *Polyspace Products for Ada Reference Guide*.

New Options to Classify Run-Time Checks

When reviewing results in the Run-Time Checks perspective, the software now provides additional options for classifying checks



example.c / Recursion / line 142 / column 15

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

Classification	Status	Justified	Comment
Not a defect	Not an Issue	✓	Value of depth cannot be negative.

```
Warning : float division by zero may occur
   Orange may be caused by the stubbed function random_int in example.c line 152 column 10
```

After you review the check, you can specify the following:

- **Classification** – Select an option to describe the seriousness of the issue.
- **Status** – Select an option to describe how you intend to address the issue.
- **Justified** – Select the check box to indicate that you have justified this check.
- **Comment** – Enter additional information about the check

The software provides pre-defined values for Classification and Status. You can also define your own statuses.

In addition to reviewing checks through the user interface, you can place comments in your code that highlight and categorize checks identified in previous verifications. The software displays the information that you provide within your code comments, and marks the checks as **Justified**.

For more information, see “Reviewing and Commenting Checks” in the *Polyspace Products for Ada User’s Guide*.

Compatibility Considerations. The syntax for code comments has changed to reflect the new options for categorizing checks.

The syntax for run-time checks is now:

```
-- polyspace<RTE:RTE1 : [Classification] : [Status] > [Comment]
```

If you placed comments in your code using the previous syntax, the comments will still appear in your results, but the text may be displayed in different columns.

For more information, see "Syntax for Run-Time Errors" in the *Polyspace Products for Ada User's Guide*.

Permissiveness on File and Folder Names

Polyspace software now allows space characters in the names of Projects, source files, and folders, as well as in option arguments.

In addition, multiple source files with the same name are now allowed.

Note Non-ASCII characters in file names are not supported.

Default Target Processor

The default setting of the Target processor type (`-target-processor`) option has changed from `SPARC` to `i386`.

Compatibility Considerations. If you launch verifications without specifying a value for this option, the default value has changed. Therefore, your verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Operating System Support

Added support for the Windows 7 operating system.

Solaris operating system is no longer supported for new installations.

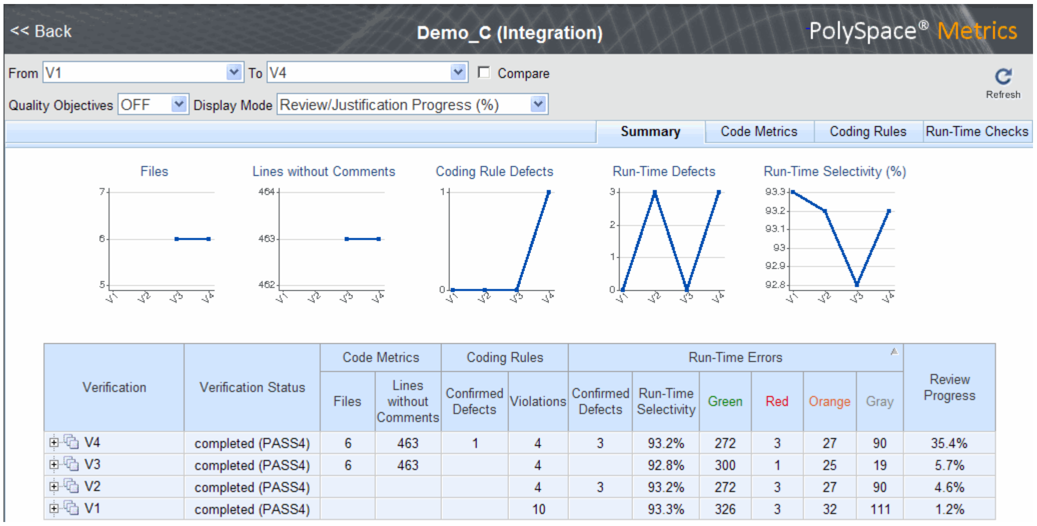
For more information, see the *Polyspace Installation Guide*.

Polyspace Server for Ada Product

Polyspace Metrics Web Interface

A web-based tool for software development managers, quality assurance engineers, and software developers, which allows you to do the following in software projects:

- Evaluate software quality metrics
- Monitor the variation of code metrics, and run-time checks through the lifecycle of a project
- View defect numbers, run-time reliability of the software, review progress, and the status of the code with respect to software quality objectives.



From V1 To V4 Compare

Quality Objectives OFF Display Mode Review/Justification Progress (%)

Verification	Verification Status	Code Metrics		Coding Rules		Run-Time Errors						Review Progress
		Files	Lines without Comments	Confirmed Defects	Violations	Confirmed Defects	Run-Time Selectivity	Green	Red	Orange	Gray	
V4	completed (PASS4)	6	463	1	4	3	93.2%	272	3	27	90	35.4%
V3	completed (PASS4)	6	463		4		92.8%	300	1	25	19	5.7%
V2	completed (PASS4)				4	3	93.2%	272	3	27	90	4.6%
V1	completed (PASS4)				10		93.3%	326	3	32	111	1.2%

In addition, if you have the Polyspace Client for Ada product installed on your computer, you can drill down to run-time checks in the Polyspace verification environment. You can review these run-time checks and, if required, classify these checks as defects.

For more information, see Software Quality with Polyspace Metrics in the *Polyspace Products for Ada User's Guide*.

Automatic Verification

Configure verifications to start automatically and periodically, for example, at a specific time every night. At the end of each verification, the software stores results in a results repository and updates the metrics for your software project. You can also configure the software to send you an email at the end of the verification. This email contains links to results, compilation errors, run-time errors, or processing errors.

For more information, see Specifying Automatic Verification in the *Polyspace Products for Ada User's Guide*.

Operating System Support

Added support for the Windows 7 operating system.

Solaris operating system is no longer supported for new installations.

For more information, see the *Polyspace Installation Guide*.

Version 5.6 (R2010b) Polyspace Model Link Products

This table summarizes what's new in V5.6 (R2010b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Model Link SL Bug Reports Polyspace Model Link TL Bug Reports Polyspace UML Link RH Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace Model Link SL Product" on page 77
- "Polyspace UML Link RH Product" on page 78

Polyspace Model Link SL Product

Data Range Management

Improved DRS generation in the Polyspace Model Link SL product using the Embedded Coder™ `codeInfo` feature.

DRS generation can now:

- Locate input ports, and gather min/max data from the MATLAB workspace or blocks in the model to use as constraints.
- Locate output ports, and gather min/max data from the MATLAB workspace or blocks in the model to use as properties to be proven.

For more information, see the *Polyspace Model Link Products User's Guide*.

Verification Options Set by Default

The following options are no longer set by default when you launch a verification.

- `-ignore-float-rounding`
- `-allow-ptr-arith-on-struct`

Compatibility Considerations. Verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Changes to verification results will be more significant when compared to release R2008b or earlier, and less significant with R2009a or later.

Simulink Software Support

Added support for Simulink Version 7.6 (R2010b).

Polyspace UML Link RH Product

Rhapsody Support

Added support for Telelogic® Rhapsody® Version 7.4.

Version 7.2 (R2010a) Polyspace for C/C++ Products

This table summarizes what's new in V7.2 (R2010a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for C/C++ Product" on page 79
- "Polyspace® Server for C/C++ Product" on page 103

Polyspace Client for C/C++ Product

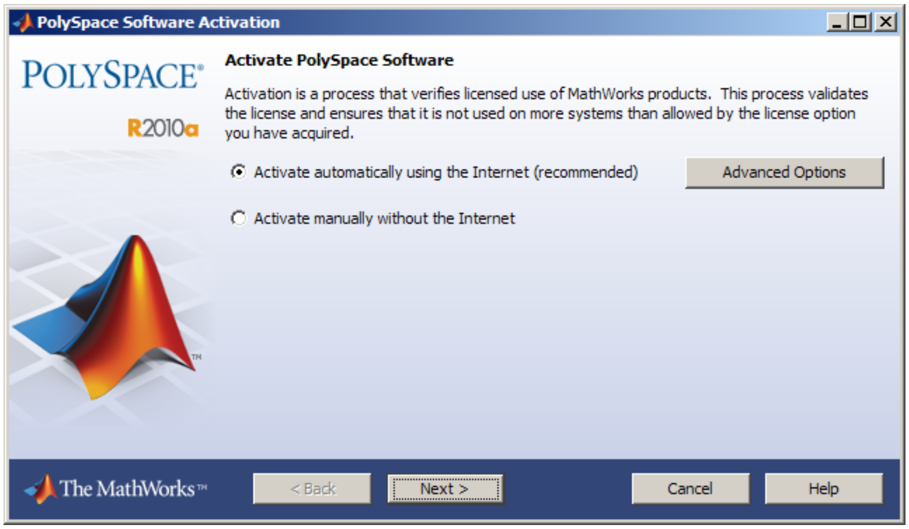
License Activation

Polyspace products now support the MathWorks software activation mechanism.

Activation is a process that verifies licensed use of MathWorks products. The process validates your product licenses and ensures that they are used correctly. You must complete the activation process before you can use Polyspace software.

Note If you are using Designated Computer (Individual) licenses, you must activate the license for each Polyspace system individually. However, if you are using Concurrent licenses for multiple Polyspace systems, you do not need to activate each Polyspace system. You activate the license once (for the FLEXnet license server), then provide license files for each Polyspace system.

The easiest way to activate the software is to log in to your MathWorks Account during installation. At the end of the installation process, the Polyspace Software Activation dialog box opens.



Activate PolySpace Software

Activation is a process that verifies licensed use of MathWorks products. This process validates the license and ensures that it is not used on more systems than allowed by the license option you have acquired.

[✓ Activate automatically using the Internet (recommended)]

Activate manually without the Internet

Follow the prompts in the Polyspace Software Activation dialog box to complete the activation process.

If you do not have a MathWorks account, you can create one during the activation process. To create an account, you must have an Activation Key, which identifies the license you want to install and activate.

If your Polyspace system is not connected to the internet, you can access the MathWorks License Center on a computer with internet access, activate your license, and download a license file for transfer to your Polyspace system. If you do not have access to a computer with an Internet connection, contact Customer Support.

For more information on how to activate your software, see “Activating Polyspace Software”in the *Polyspace Installation Guide*.

For more information on software activation, including frequently asked questions, refer to the MathWorks Web site:
www.mathworks.com/support/activation/polyspace.html

MISRA C++ Checker

Polyspace software can now analyze your C++ code to check compliance with the MISRA C++ coding standard.

The Polyspace MISRA C++ checker provides messages when MISRA C++ rules are not respected. Most messages are reported during the compile phase of a verification.

The MISRA C++ checker can check 167 of the 183 statically enforceable MISRA C++ coding rules.

Note The Polyspace MISRA C++ checker is based on MISRA C++:2008 – “Guidelines for the use of the C++ language in critical systems." For more information on these coding standards, see http://www.misra-cpp.com.

For more information, see “Checking Coding Rules”, in the *Polyspace Products for C++ User’s Guide*.

Source Code Comments

Polyspace software now allows you to place comments in your code that provide information about known coding rule violations and run-time errors. You can use these comments to

- Hide or highlight known coding rule violations.
- Highlight and categorize previously identified run-time errors.

This information can then make the review process quicker and easier by allowing you to focus on new coding rule violations and run-time errors. .

When you review verification results, the Viewer displays comments on individual checks. You can then skip these commented checks, or simply use them as additional information during your review.

The coding rules log in the Launcher displays comments regarding coding rules. You can use these comments to filter out commented violations from the results, or simply to provide additional information on specific violations.

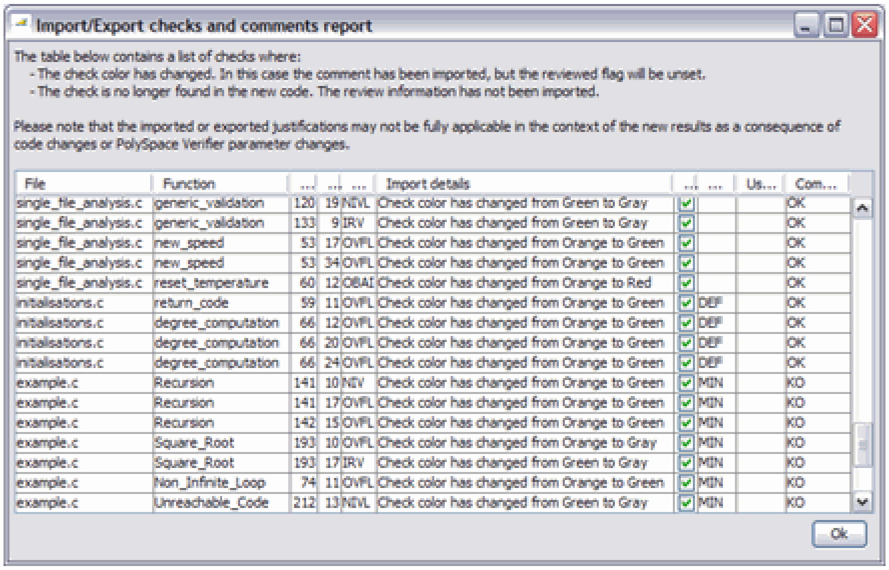
For more information, see “Highlighting Known Coding Rule Violations and Run-Time Errors” in the *Polyspace Products for C User’s Guide* or *Polyspace Products for C++ User’s Guide*.

Importing Review Comments

New Import/Export checks and comments report allows you to you to compare the source code and verification results from a previous verification to the current verification, and highlights differences in the results.

Importing review comments from a previous verification can be extremely useful, since it allows you to avoid reviewing checks twice, and to compare verification results over time. However, if your code has changed since the previous verification, or if you have upgraded to a new version of the software, the imported comments may not be applicable to your current results. For example, the color of a check may have changed, or the justification for an orange check may no longer be relevant to the current code.

Use the Import/Export checks and comments report to highlight these differences, and focus on unreviewed results.



The table below contains a list of checks where:
- The check color has changed. In this case the comment has been imported, but the reviewed flag will be unset.
- The check is no longer found in the new code. The review information has not been imported.

Please note that the imported or exported justifications may not be fully applicable in the context of the new results as a consequence of code changes or PolySpace Verifier parameter changes.

File	Function	...	...	...	Import details	...	...	Us...	Com...
single_file_analysis.c	generic_validation	120	19	NIVL	Check color has changed from Green to Gray	✓			OK
single_file_analysis.c	generic_validation	133	9	IRV	Check color has changed from Green to Gray	✓			OK
single_file_analysis.c	new_speed	53	17	OVFL	Check color has changed from Orange to Green	✓			OK
single_file_analysis.c	new_speed	53	34	OVFL	Check color has changed from Orange to Green	✓			OK
single_file_analysis.c	reset_temperature	60	12	OBAI	Check color has changed from Orange to Red	✓			OK
initialisations.c	return_code	59	11	OVFL	Check color has changed from Orange to Green	✓	DEF		OK
initialisations.c	degree_computation	66	12	OVFL	Check color has changed from Orange to Green	✓	DEF		OK
initialisations.c	degree_computation	66	20	OVFL	Check color has changed from Orange to Green	✓	DEF		OK
initialisations.c	degree_computation	66	24	OVFL	Check color has changed from Orange to Green	✓	DEF		OK
example.c	Recursion	141	10	NIV	Check color has changed from Orange to Green	✓	MIN		KO
example.c	Recursion	141	17	OVFL	Check color has changed from Orange to Green	✓	MIN		KO
example.c	Recursion	142	15	OVFL	Check color has changed from Orange to Green	✓	MIN		KO
example.c	Square_Root	193	10	OVFL	Check color has changed from Orange to Gray	✓	MIN		KO
example.c	Square_Root	193	17	IRV	Check color has changed from Green to Gray	✓	MIN		KO
example.c	Non_Infinite_Loop	74	11	OVFL	Check color has changed from Orange to Green	✓	MIN		KO
example.c	Unreachable_Code	212	13	NIVL	Check color has changed from Green to Gray	✓	MIN		KO

For more information, see "Importing and Exporting Review Comments" in the *Polyspace Products for C User's Guide*.

Compatibility Considerations. In previous releases, when you specified the option `-keep-all-files`, it was possible to add comments to the results for a specific verification level (for example, `pass2`) , and then import them into another set of results (for example `pass4`) in the same results folder.

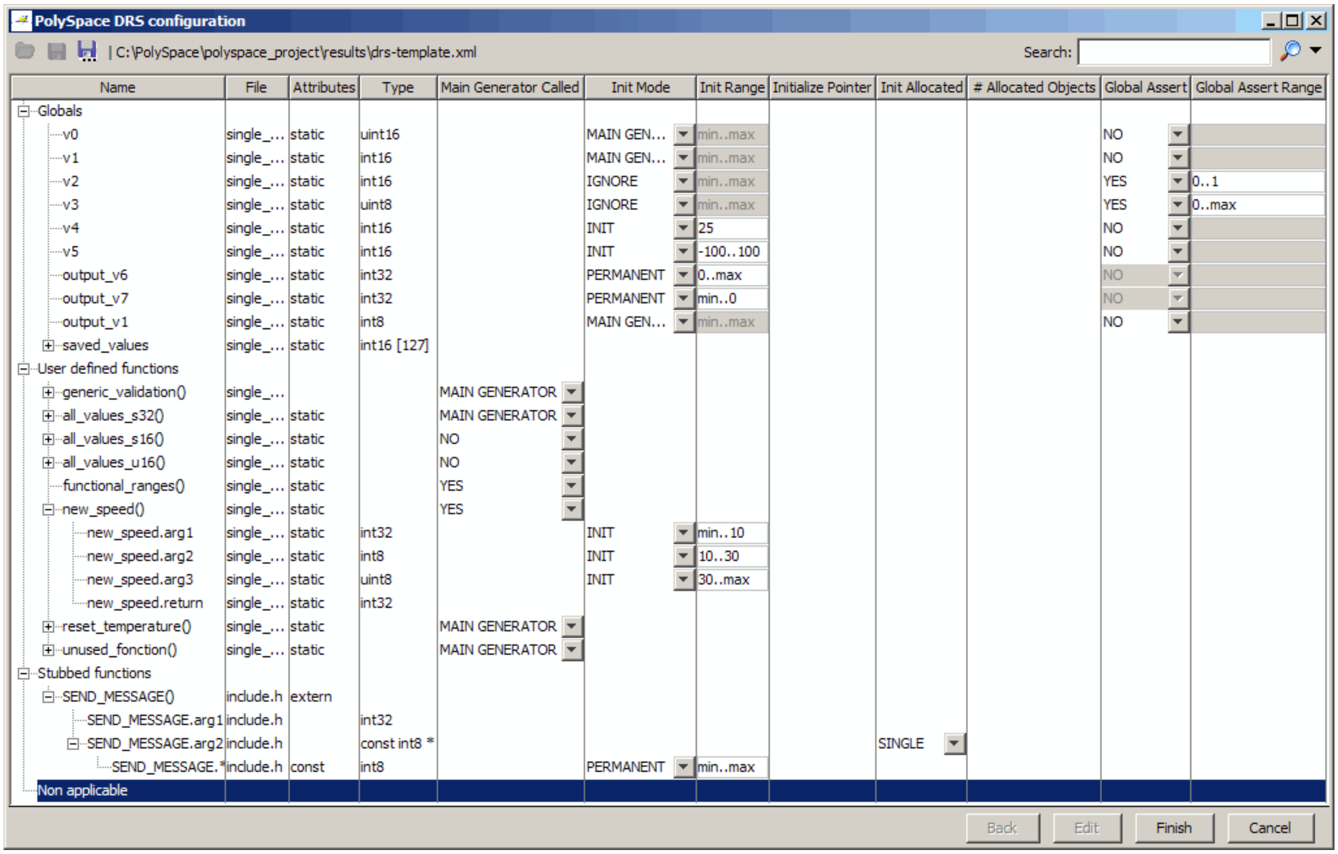
This is no longer possible in R2010a.

Data Range Specifications (DRS) Enhancements

Enhanced Data Range Specifications, including new format and workflow.

The Polyspace Data Range Specifications (DRS) feature now allows you to set constraints on data ranges using a new graphical user interface. When you enable the DRS feature, Polyspace software analyzes the files in your project, and generate a DRS template containing all the global variables, user defined functions, and stub functions for which you can specify data ranges.

To specify data ranges, you then edit this template using the Polyspace DRS configuration interface.


PolySpace DRS configuration
| C:\PolySpace\polyspace_project\results\drs-template.xml
Search:
Back
Edit
Finish
Cancel

Name	File	Attributes	Type	Main Generator Called	Init Mode	Init Range	Initialize Pointer	Init Allocated	# Allocated Objects	Global Assert	Global Assert Range
Globals											
v0	single_...	static	uint16		MAIN GEN...	min..max				NO	
v1	single_...	static	int16		MAIN GEN...	min..max				NO	
v2	single_...	static	int16		IGNORE	min..max				YES	0..1
v3	single_...	static	uint8		IGNORE	min..max				YES	0..max
v4	single_...	static	int16		INIT	25				NO	
v5	single_...	static	int16		INIT	-100..100				NO	
output_v6	single_...	static	int32		PERMANENT	0..max				NO	
output_v7	single_...	static	int32		PERMANENT	min..0				NO	
output_v1	single_...	static	int8		MAIN GEN...	min..max				NO	
saved_values	single_...	static	int16 [127]								
User defined functions											
generic_validation()	single_...			MAIN GENERATOR							
all_values_s32()	single_...	static		MAIN GENERATOR							
all_values_s16()	single_...	static		NO							
all_values_u16()	single_...	static		NO							
functional_ranges()	single_...	static		YES							
new_speed()	single_...	static		YES							
new_speed.arg1	single_...	static	int32		INIT	min..10					
new_speed.arg2	single_...	static	int8		INIT	10..30					
new_speed.arg3	single_...	static	uint8		INIT	30..max					
new_speed.return	single_...	static	int32								
reset_temperature()	single_...	static		MAIN GENERATOR							
unused_fonction()	single_...	static		MAIN GENERATOR							
Stubbed functions											
SEND_MESSAGE()	include.h	extern									
SEND_MESSAGE.arg1	include.h		int32								
SEND_MESSAGE.arg2	include.h		const int8 *					SINGLE			
SEND_MESSAGE.*	include.h	const	int8		PERMANENT	min..max					
Non applicable											

In addition, the DRS feature now allows you to specify constraints for additional types of data, including:

- Input parameters for user-defined functions called by the main generator
- Static variables
- Pointers (C only)

For more information, see "Specifying Data Ranges for Variables and Functions (Contextual Verification)" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Compatibility Considerations. Symbols ranged by DRS (`init`, `permanent` or `globalassert` mode) are no longer ignored by the main-generator. This can lead to differences in values and colors, for example full range instead of 0, or orange instead of green.

Pointer Information in the Viewer

Enhanced ToolTips in the Viewer now display pointer information, in addition to data ranges.

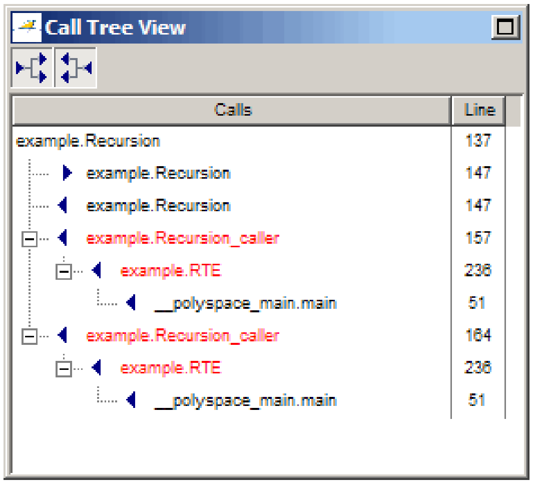
The software now provides, through tooltip messages, useful information about pointers to variables or functions. You see this information in the source code view when you place your cursor over a pointer, dereference character, function call, or function declaration. In addition, if you click a pointer check, dereference character, function call, or function declaration, the software displays pointer information in the selected check view.

For more information, see "Using Pointer Information in Run-Time Checks Perspective" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Enhanced Call Tree View and Variables View (Data Dictionary)

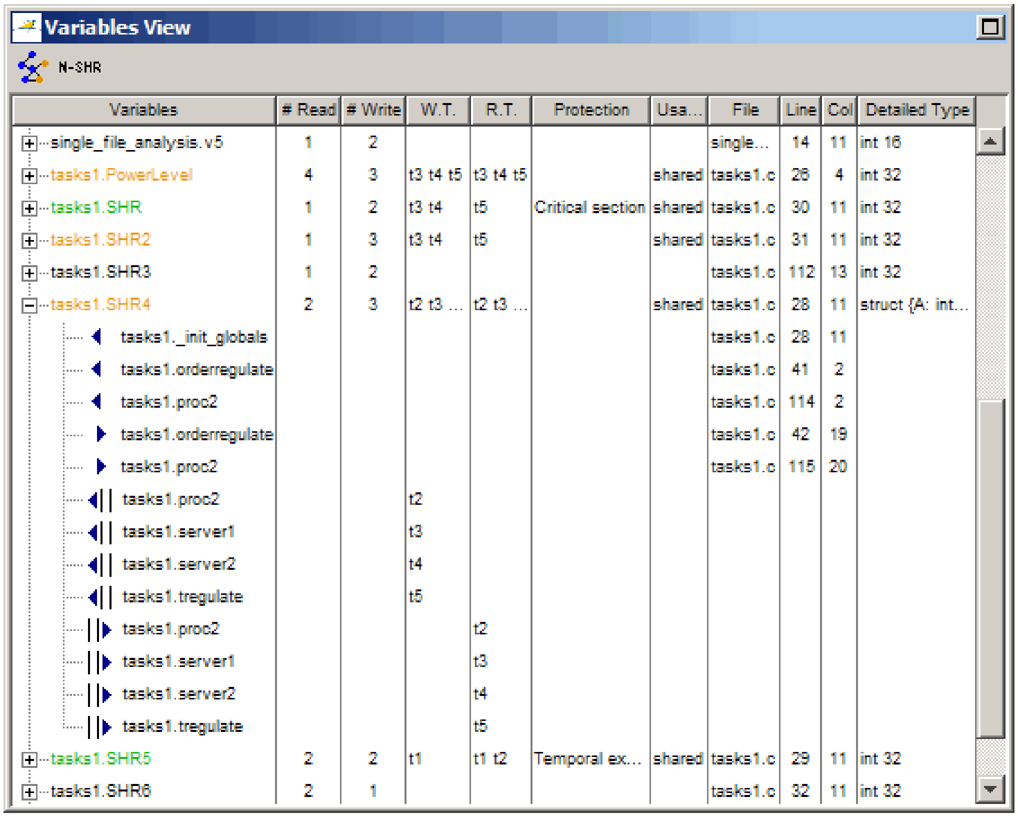
Enhanced user interface of the Call Tree View and Variables View improves navigation and usability.

In the Call Tree View, you can now double click any function call to go directly to the function definition.



Calls	Line
example.Recursion	137
example.Recursion	147
example.Recursion	147
example.Recursion_caller	157
example.RTE	236
__polyspace_main.main	51
example.Recursion_caller	164
example.RTE	236
__polyspace_main.main	51

In the Variables View, you can now right-click a variable to show legend information, and can open the concurrent access graph for a variable directly from the Variables View.



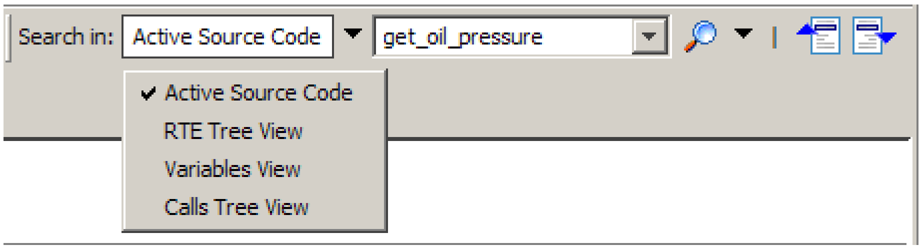
Variables	# Read	# Write	W.T.	R.T.	Protection	Usa...	File	Line	Col	Detailed Type
single_file_analysis.v5	1	2					single...	14	11	int 16
tasks1.PowerLevel	4	3	t3 t4 t5	t3 t4 t5		shared	tasks1.c	26	4	int 32
tasks1.SHR	1	2	t3 t4	t5	Critical section	shared	tasks1.c	30	11	int 32
tasks1.SHR2	1	3	t3 t4	t5		shared	tasks1.c	31	11	int 32
tasks1.SHR3	1	2					tasks1.c	112	13	int 32
tasks1.SHR4	2	3	t2 t3 ...	t2 t3 ...		shared	tasks1.c	28	11	struct {A: int...
◀ tasks1._init_globals							tasks1.c	28	11	
◀ tasks1.orderregulate							tasks1.c	41	2	
◀ tasks1.proc2							tasks1.c	114	2	
▶ tasks1.orderregulate							tasks1.c	42	19	
▶ tasks1.proc2							tasks1.c	115	20	
◀|| tasks1.proc2			t2							
◀|| tasks1.server1			t3							
◀|| tasks1.server2			t4							
◀|| tasks1.tregulate			t5							
||▶ tasks1.proc2				t2						
||▶ tasks1.server1				t3						
||▶ tasks1.server2				t4						
||▶ tasks1.tregulate				t5						
tasks1.SHR5	2	2	t1	t1 t2	Temporal ex...	shared	tasks1.c	29	11	int 32
tasks1.SHR6	2	1					tasks1.c	32	11	int 32

For more information, see "Exploring the Run-Time Checks Perspective" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Enhanced Search Function in Viewer

Enhanced Search feature in the Viewer improves navigation in your results.

The Viewer toolbar now contains a Search interface. This allows you to quickly enter search terms, specify search options, and set the scope for your search.



For more information, see "Exploring the Run-Time Checks Perspective" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Filtering Orange Checks in Viewer (C only)

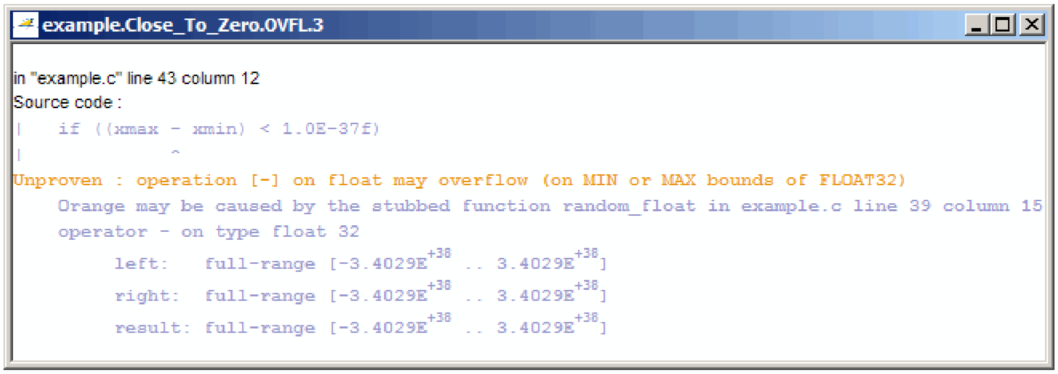
Polyspace verification now identifies orange checks caused by input data. The software provides additional information on these orange checks, and allows you to hide them in the Viewer.

Note Although this type of orange check could reveal a bug, they usually do not.

Verification can identify orange checks caused by:

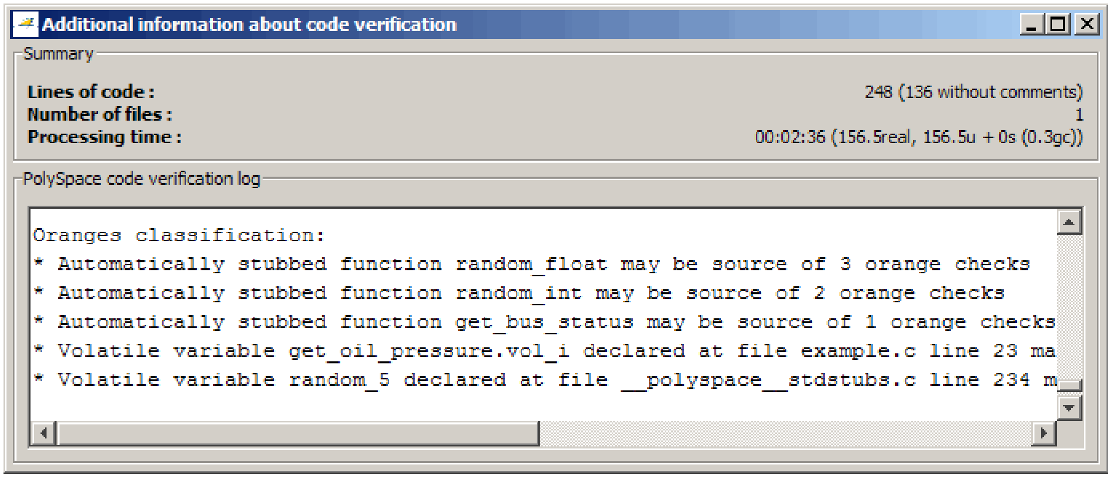
- Stubs
- Main-generator calls
- Volatile variables
- Extern variables
- Absolute address

When the software identifies this type of orange check, the Viewer provides information on its cause.



```
in "example.c" line 43 column 12
Source code :
|    if ((xmax - xmin) < 1.0E-37f)
|               ^
Unproven : operation [-] on float may overflow (on MIN or MAX bounds of FLOAT32)
    Orange may be caused by the stubbed function random_float in example.c line 39 column 15
    operator - on type float 32
         left:   full-range [-3.4029E+38 .. 3.4029E+38]
         right:  full-range [-3.4029E+38 .. 3.4029E+38]
         result: full-range [-3.4029E+38 .. 3.4029E+38]
```

The Polyspace code verification log file also lists possible sources of imprecision for orange checks.



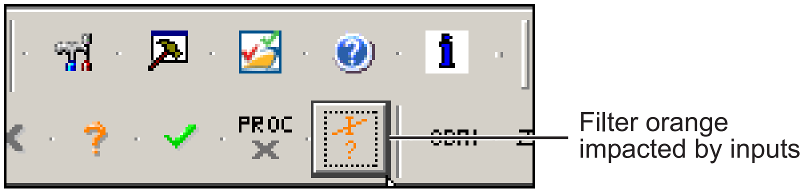
Additional information about code verification
Summary

Lines of code :	248 (136 without comments)
Number of files :	1
Processing time :	00:02:36 (156.5real, 156.5u + 0s (0.3gc))

PolySpace code verification log

```
Oranges classification:
* Automatically stubbed function random_float may be source of 3 orange checks
* Automatically stubbed function random_int may be source of 2 orange checks
* Automatically stubbed function get_bus_status may be source of 1 orange checks
* Volatile variable get_oil_pressure.vol_i declared at file example.c line 23 ma
* Volatile variable random_5 declared at file __polyspace__stdstubs.c line 234 m
```

In addition, you can now hide these types of orange checks in the Viewer. When using Expert mode, click the filter button to hide oranges impacted by input data.



For more information, see “Working with Orange Checks Caused by Input Data” in the *Polyspace Products for C User’s Guide* or *Polyspace Products for C++ User’s Guide*.

Methodological Assistant Enhancements

Enhanced Methodological Assistant in the Viewer.

The Methodological Assistant now allows you to define either a minimum percentage of orange checks to review, or a specific number of orange checks to review. This makes it easier to set specific quality criteria for your code at each level of review.

In addition, the Methodological Assistant now presents checks in a more logical order. Checks that are most likely to reveal bugs appear first, while non-useful checks no longer appear.

The new order of checks is:

1 All red checks (an error always occurs)

2 Orange checks known to produce errors in some situations (dark orange). For example, red for one call to a procedure and green for another.

3 Some gray checks (UNR checks)

4 Other orange checks (depending on the methodology and criterion level)

Most gray checks no longer appear in the Methodological Assistant, since reviewing many gray checks that occur after a red check is not useful. Only UNR checks that are not nested within dead code blocks appear in assistant mode.

Compatibility Considerations. The number of checks presented for review in Assistant mode is different than in previous releases, since most gray checks no longer appear. In addition, the order in which you review checks is different.

Class Analyzer Enhancements for C++

Enhanced class analyzer can analyze a file with more than one class.

Unit-by-unit verifications can now verify files containing more than one class. Every class and function out of class contained in such files is now verified.

For more information, see "Polyspace Class Analyzer" in the *Polyspace Products for C++ User's Guide*.

Compatibility Considerations. In `-unit-by-unit` mode, files that previously were not verified because they contained more than one class are now verified.

Change to Time Format in Log File

The time format reported in the log file has been updated to provide more information.

Example of new line (R2010a and later):

```
User time for polyspace-c:  00:02:24 (144.6real, 144.6u + 0s
(0.3gc))
```

Example of old line (R2009b and earlier):

```
User time for rte-kernel:  4684.4real, 4319.2u + 324.6s (0.3gc)
```

Compatibility Considerations. The new time format can impact some scripts that summarize information from the log file.

Merging of `OVFL` and `UNFL` Checks

Overflow (`OVFL`) and underflow (`UNFL`) checks have been merged into a single `OVFL` check. This reduces the number of orange checks you need to review, while continuing to provide the same information.

For red and orange checks, the check message provides the bounds that cause the overflow.

Compatibility Considerations. The Selectivity rate of your results may change when compared to previous versions of the software. Underflows and overflows are now identified as a single check, so the Selectivity will decrease if the checks were green (2 green checks become 1 green), but will increase if the checks were both orange (2 orange checks become 1 orange).

Improved `UNR` Checks

Enhanced unreachable code (UNR) checks now provide additional information to help you understand the results. UNR checks now include information on:

- Localization of condition
- Type of condition
- End of block localization

For example:

```
// UNR (unreachable code) => UNR (unreachable code)   \
(end of block at line YYY)

// UNR (unreachable code) => UNR (unreachable code)   \
(condition at line XXX, column AAA) ?
```

In addition, verification now reports new `UNR` checks on:

- unreachable statements after `return`, `break`, `goto`, and `continue` statements.
- `if` statements when the `if` condition is always true and if there is no `else` statement.

For more information on these new checks, see "Changes to Verification Results" on page 93.

Compatibility Considerations. The number of checks in your verification results may change due to the new `UNR` checks.

Changes to Verification Results

Compatibility Considerations. Verification results may change when compared to previous versions of the software. Some checks may change color, and the Selectivity rate of your results may change.

Refer to the following sections for information on the specific changes.

Merging of OVFL and UNFL Checks. Overflow (OVFL) and underflow (UNFL) checks have been merged into a single OVFL check. This reduces the number of orange checks you need to review, while continuing to provide the same information.

For red and orange checks, the check message provides the bounds that cause the overflow.

The Selectivity rate of your results may change when compared to previous versions of the software.

New Gray (`UNR`) Checks on `return`, `break`, `goto`, and `continue` Statements. Verification now reports gray `UNR` checks on unreachable statements after `return`, `break`, `goto`, and `continue` statements.

For example:

```
switch (counter) {
 case 0:
  counter = 0;
  break;
case 1:
 counter = 2;
 break;
  counter = 2; /* unreachable code ! */
  break;
```

The number of checks in your verification results may increase due to these new UNR checks.

New Gray (`UNR`) Check on If Statement Without Else. Verification now reports a gray `UNR` check on an `if` statement when the `if` condition is always true and if there is no `else` statement.

This allows you to find `if` branches that are always reachable, even when there is no `else`.

For example:

```
if (true())  // UNR if-condition always evaluates to true
```

```
    {
     // ...
    }
```

The number of checks in your verification results may increase due to new UNR checks.

Nested Gray (UNR) Checks No Longer Appear in Reports. Nested UNR checks in unreachable blocks no longer appear in the Methodological Assistant, or in generated reports.

The number of checks in generated reports may decrease due to elimination of these checks..

Dead Code on Else Branch. Verification now reports gray `UNR` checks on empty branches.

For example:

```
void fct (void)
{
 int a = 1;
 if (a){
  a++;
 }
 else // ==> Now gray UNR
 {
 // dummy
 }
}
```

The number of checks in your verification results may increase due to the new UNR check on empty branches.

Data Ranges for Fields of Structures (C). Symbols ranged by DRS (`init`, `permanent` or `globalassert` mode) are now considered by the main-generator.

In previous releases, if DRS provided ranges for some fields of a structure, the other fields (not ranged by DRS) were not initialized by the main-generator, and therefore had an initial value of 0.

For example:

```
// DRS: s.x 0 10 init

struct { int x; int y; } s;
int foo(void)
{
  return s.y; // y value: 0, full-range expected
}
```

Symbols ranged by DRS are no longer ignored by the main-generator. This can lead to differences in values and colors, for example full range instead of 0, or orange instead of green.

Functions Called Before Main in Unit-by-unit Verification (C++). The behavior of the option `-function-called-before-main` has changed for unit-by-unit verifications of C++ code.

When you set the option `-function-called-before-main` in unit-by-unit mode:

- If the `init` function is an out of class function, it is called at the beginning of the generated main (before calls to constructors).
- If the `init` function is a method, it is called after all constructor calls of the corresponding class.

In previous releases, the `init` function was always called after constructor calls for each class.

Verification results may change when compared to previous versions of the software, due to changes in the call sequence.

Main Generator Initialization of Function Pointers. The main-generator now initializes function pointers with default-mode stubs instead of pure stubs.

In previous releases, the main-generator initialized function pointers with pointers to pure functions.

This change may lead to differences in the color of checks in your results. For example:

```
int x;
  s->fptr(&x);
  read(x); // LNIV red with 9b -> orange with 10a
```

OVFL Check on Array Index Removed. In previous releases, verification reported an overflow (OVFL) check on pointer/array dereference. However, this overflow never occurred if there was an OBAI problem first. Therefore, the check was not useful.

In R2010a, the OVFL check no longer appears on array index, the check has been merged into the OBAI check.

For example, in the following code there is no OVFL check on the array index.

```
int main(void)
{
   volatile int i,x;
   int tab[10];

   x = tab[i];

}
```

The Selectivity of your results may change when compared to previous versions of the software. The OVFL check on array access has been merged into the OBAI check, so there are fewer checks reported. Selectivity will increase if the overflow check was orange, but will decrease if the OVFL check was green.

IDP Check on Local Member Access Removed (C++). Verification no longer reports an IDP check on local member access.

In previous releases, verification reported an IDP check. This IDP appeared on the "`.`" when accessing the field of an object returned by copy construction.

For example:

```
struct C {
    C(const C&c1) { k =c1.k; }
    C() { k = 0 ;}
    int k ;
  } ;

  C g() {
    C ret ;
    ret.k = 2 ; // IDP on "." here
    return ret;
  }

  int main() {
    C c = g() ;
  }
```

However, this check was caused by an internal pointer and was not useful.

The Selectivity of your results may change when compared to previous versions of the software.

OBAI Check on Dynamic Initialization of Array Removed (C++). Verification no longer reports an OBAI check on dynamic initialization of array.

In previous releases, verification reported an OBAI check. The OBAI check appeared on dynamic initialization of array with an aggregate.

For example:

```
nt main(void)
{
  float tab[] = // extra green obai check
    {
      4.3,
      0.0F
    };

  return 0;
}
```

However, this check was caused by an internal translation and was not useful.

The Selectivity of your results may change when compared to previous versions of the software.

Duplicate Checks in `For/While` Loops Removed. Verification no longer reports duplicate checks in condition expression of `for` and `while` loops.

Any duplicate checks on a loop condition are now merged in a single check, except when condition expression is complex.

Due to the reduction in the number of checks, the selectivity of your results may change when compared to previous versions of the software.

`malloc(0)` Limitation Removed. Verification no longer has a limitation when `malloc(0)` returns a null pointer.

In previous releases, verification reported a green check on the following code:

```
assert(malloc(0) == NULL) ;
```

However, this construction could fail. The software now correctly verifies this construction.

Verification results may change when compared to previous versions of the software.

Change in OOP on Deletion of Null Pointer (C++). Verification no longer reports a red OOP check when deleting a null pointer.

In previous releases, verification reported a red OOP check on the following code:

```
struct A {
  virtual void f() { }
  ~A() { }
} ;

int main() {
  A* pa ;
  if (0) pa = (A*) 0Xfff;
```

```
    pa = 0;
    delete pa ; // red OOP
  }
```

However, calling "delete" on a null pointer is allowed. The red OOP when deleting a null pointer is now gray.

Verification results may change when compared to previous versions of the software.

Change to IDP Check When Accessing a Field of an Inherited Class (C). Verification no longer reports a red IDP check when accessing a field of an inherited class with an mcpu target.

In previous releases, verification reported a red IDP check.

For example:

```
struct Val {
 int val;
};

struct Left : virtual Val {
 int left;
 virtual int get_left() { return left; } // polymorphic:yes
};

struct Right : virtual Val {
 int right;
 virtual int get_right() { return right; }
};

struct S : Left, Right {      // multiple:yes
};

S s = S();
Left& le = s;         // intermediate:global, reference:yes
Right& re = s;         // intermediate:global, reference:yes

int main(void){
    assert(re.val == 0);    // Unexpected red IDP
  }
```

However, this was not actually an error. The check is no longer red.

The color of the IDP check has changed when compared to previous versions of the software.

Changes to Coding Rules Checker Results

- "Compatibility Considerations" on page 101
- "MISRA-C Rule 10.1 Violations on Constant Operands" on page 101
- "MISRA-C Rule 12.5 Violation Report Improved" on page 102
- "MISRA-C Rule 7.1 Violations on File Names of Preprocessed Files" on page 102
- "MISRA-C Rule 5.4 Violations on Anonymous Structures and Unions" on page 102
- "JSF Rule AV-151 Violations on Evaluation of Constant" on page 102

Compatibility Considerations. Due to changes in the coding rules checker, the number of coding rule violations may change when compared to previous versions of the software.

Refer to the following sections for information on the specific changes.

MISRA-C Rule 10.1 Violations on Constant Operands. The MISRA-C checker no longer reports errors for rule 10.1, "The value of an expression of integer type shall not be implicitly converted to a different underlying type," for certain constructions. For example:

```
int i;
for (i = 0; i < 12; i++)
```

An integer constant that fits into the size of a `char` is now seen as a `signed char` whatever the sign of `char` (this depends on the selected target or is set by option).

If you use the options `-target powerpc` or `-default-sign-of-char unsigned`, the coding rules checker will report fewer violations of MISRA-C rule 10.1 on constant operands.

MISRA-C Rule 12.5 Violation Report Improved. The coding rules checker now reports a column number for violations of MISRA-C rule 12.5.

You may see more violations of rule 12.5, since two violations that occur on same line but in different columns are now identified separately.

MISRA-C Rule 7.1 Violations on File Names of Preprocessed Files. The coding rules checker no longer reports violations of MISRA-C rule 7.1 on the names of internal preprocessing files. These violations occurred in projects containing Japanese characters.

You may see fewer violations of rule 7.1 in MISRA reports.

MISRA-C Rule 5.4 Violations on Anonymous Structures and Unions. The coding rules checker no longer reports violations of MISRA-C rule 5.4 on anonymous struct/union fields.

You may see fewer violations of rule 5.4 in MISRA reports.

JSF Rule AV-151 Violations on Evaluation of Constant. The coding rules checker no longer reports violations of JSF rule AV-151 on internal evaluation of a constant value, for example when there is an expression in an enum list.

You may see fewer violations of rule AV-151 in JSF reports.

Enumerated Types Support

The option `-enum-type-definition` allows verification to use different base types to represent an enumerated type, depending on the enumerator values and the selected definition.

When using this option, each enum type is represented by the smallest integral type that can hold all its enumeration values.

Possible values are:

- `defined-by-standard`.
- `auto-signed-first`.
- `auto-unsigned-first`

For more information, see “Enum type definition (`-enum-type-definition`)” in the *Polyspace Products for C Reference* or *Polyspace Products for C++ Reference*.

New Target Processor Support

Added support for the `c18` 24-bit target processor (C only).

For more information, see “Predefined Target Processor Specifications” in the *Polyspace Products for C User’s Guide* or *Polyspace Products for C++ User’s Guide*.

Operating System Support

Added support for the following Linux® distributions:

- OpenSuSE 11.1
- Debian 5.x
- Ubuntu 8.04, 8.10, 9.04, and 9.10

For more information, see the *Polyspace Installation Guide*.

Polyspace Server for C/C++ Product

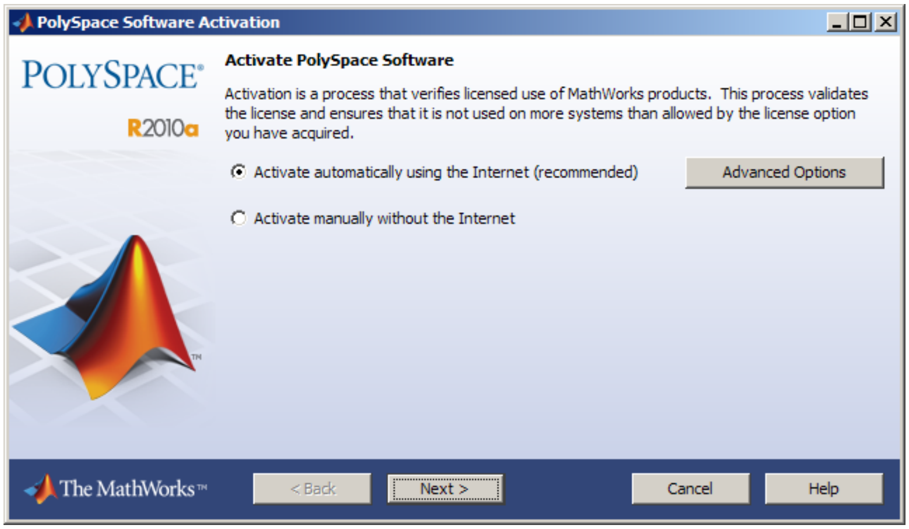
License Activation

Polyspace products now support the MathWorks software activation mechanism.

Activation is a process that verifies licensed use of MathWorks products. The process validates your product licenses and ensures that they are used correctly. You must complete the activation process before you can use Polyspace software.

Note If you are using Designated Computer (Individual) licenses, you must activate the license for each Polyspace system individually. However, if you are using Concurrent licenses for multiple Polyspace systems, you do not need to activate each Polyspace system. You activate the license once (for the FLEXnet license server), then provide license files for each Polyspace system.

The easiest way to activate the software is to log in to your MathWorks Account during installation. At the end of the installation process, the Polyspace Software Activation dialog box opens.



Activate PolySpace Software

Activation is a process that verifies licensed use of MathWorks products. This process validates the license and ensures that it is not used on more systems than allowed by the license option you have acquired.

(●) Activate automatically using the Internet (recommended)

( ) Activate manually without the Internet

Follow the prompts in the Polyspace Software Activation dialog box to complete the activation process.

If you do not have a MathWorks account, you can create one during the activation process. To create an account, you must have an Activation Key, which identifies the license you want to install and activate.

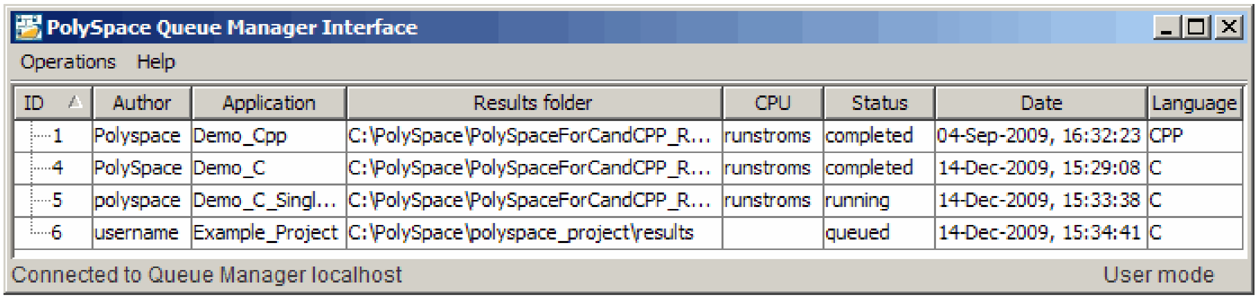
If your Polyspace system is not connected to the internet, you can access the MathWorks License Center on a computer with internet access, activate your license, and download a license file for transfer to your Polyspace system. If you do not have access to a computer with an Internet connection, contact Customer Support.

For more information on how to activate your software, see "Activating Polyspace Software"in the *Polyspace Installation Guide*.

For more information on software activation, including frequently asked questions, refer to the MathWorks Web site:
www.mathworks.com/support/activation/polyspace.html

Queue Manager Interface

The Polyspace Queue Manager Interface (Spooler) is now available on Linux machines, providing a graphical interface for managing verification jobs on the Polyspace server.



ID	Author	Application	Results folder	CPU	Status	Date	Language
1	Polyspace	Demo_Cpp	C:\PolySpace\PolySpaceForCandCPP_R...	runstroms	completed	04-Sep-2009, 16:32:23	CPP
4	PolySpace	Demo_C	C:\PolySpace\PolySpaceForCandCPP_R...	runstroms	completed	14-Dec-2009, 15:29:08	C
5	polyspace	Demo_C_Singl...	C:\PolySpace\PolySpaceForCandCPP_R...	runstroms	running	14-Dec-2009, 15:33:38	C
6	username	Example_Project	C:\PolySpace\polyspace_project\results		queued	14-Dec-2009, 15:34:41	C

Connected to Queue Manager localhost User mode

For more information, see "Managing Verification Jobs Using the Polyspace Queue Manager"in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Operating System Support

Added support for the following Linux distributions:

- OpenSuSE 11.1
- Debian 5.x
- Ubuntu 8.04, 8.10, 9.04, and 9.10

For more information, see the *Polyspace Installation Guide*.

Version 5.5 (R2010a) Polyspace for Ada and Model Link Products

This table summarizes what's new in V5.5 (R2010a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports Polyspace Model Link SL Bug Reports Polyspace Model Link TL Bug Reports Polyspace UML Link RH Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for Ada Product" on page 106
- "Polyspace® Server for Ada Product" on page 109
- "Polyspace Model Link SL Product" on page 111

Polyspace Client for Ada Product

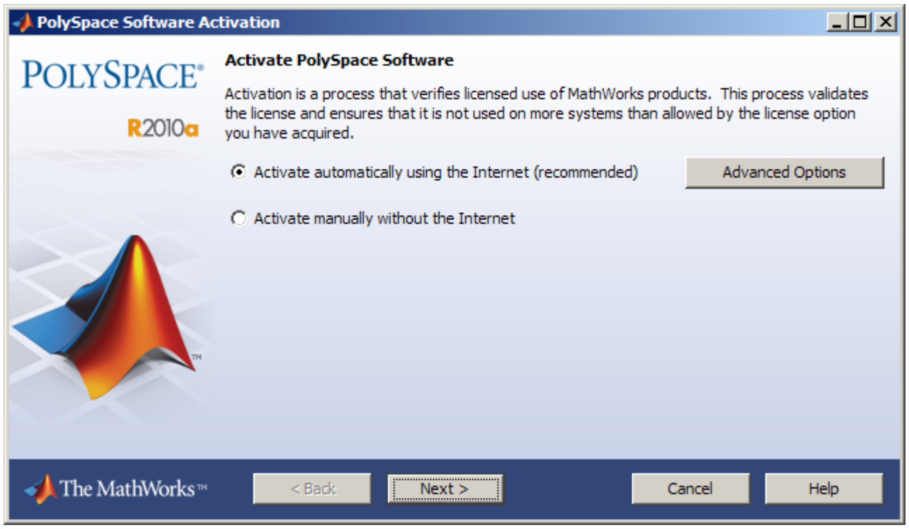
License Activation

Polyspace products now support the MathWorks software activation mechanism.

Activation is a process that verifies licensed use of MathWorks products. The process validates your product licenses and ensures that they are used correctly. You must complete the activation process before you can use Polyspace software.

Note If you are using Designated Computer (Individual) licenses, you must activate the license for each Polyspace system individually. However, if you are using Concurrent licenses for multiple Polyspace systems, you do not need to activate each Polyspace system. You activate the license once (for the FLEXnet license server), then provide license files for each Polyspace system.

The easiest way to activate the software is to log in to your MathWorks Account during installation. At the end of the installation process, the Polyspace Software Activation dialog box opens.



Activate PolySpace Software

Activation is a process that verifies licensed use of MathWorks products. This process validates the license and ensures that it is not used on more systems than allowed by the license option you have acquired.

(●) Activate automatically using the Internet (recommended)

( ) Activate manually without the Internet

Follow the prompts in the Polyspace Software Activation dialog box to complete the activation process.

If you do not have a MathWorks account, you can create one during the activation process. To create an account, you must have an Activation Key, which identifies the license you want to install and activate.

If your Polyspace system is not connected to the internet, you can access the MathWorks License Center on a computer with internet access, activate your license, and download a license file for transfer to your Polyspace system. If you do not have access to a computer with an Internet connection, contact Customer Support.

For more information on how to activate your software, see "Activating Polyspace Software"in the *Polyspace Installation Guide*.

For more information on software activation, including frequently asked questions, refer to the MathWorks Web site:
www.mathworks.com/support/activation/polyspace.html

Source Code Comment Support

Polyspace software now allows you to place comments in your code that provide information about known run-time errors. You can use these comments to highlight and categorize previously identified run-time errors. This information can then make the review process quicker and easier.

When you review verification results, the Viewer displays comments on individual checks. You can then skip these commented checks during the review process, or simply use them as additional information during your review.

For more information, see "Highlighting Known Run-Time Errors" in the *Polyspace Products for Ada User's Guide*.

Eclipse Integration

Polyspace integration with the Eclipse IDE, Version 3.4 and 3.5.

The Polyspace Client for Ada can be integrated with the Eclipse™ Integrated Development Environment through the Polyspace plug-in for Eclipse IDE.

This plug-in provides Polyspace source code verification and bug detection functionality for source code developed within Eclipse IDE. Features include the following:

- A contextual menu that allows you to launch a verification of one or more files.
- Views in the Eclipse editor that allow you to set verification parameters and monitor verification progress.

For more information, see “Using Polyspace Software in the Eclipse™ IDE” in the *Polyspace Products for Ada User’s Guide*.

Operating System Support

Added support for the following Linux distributions:

- OpenSuSE 11.1
- Debian 5.x
- Ubuntu 8.04, 8.10, 9.04, and 9.10

For more information, see the *Polyspace Installation Guide*.

Polyspace Server for Ada Product

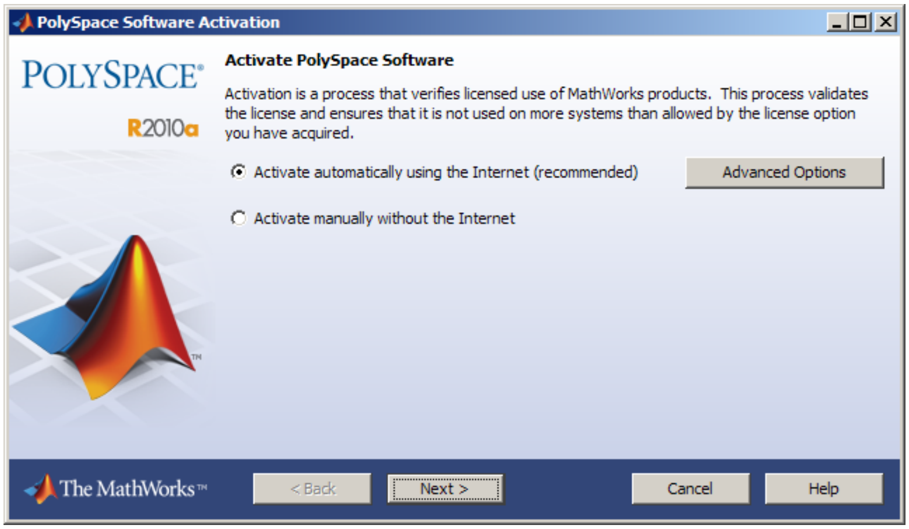
License Activation

Polyspace products now support the MathWorks software activation mechanism.

Activation is a process that verifies licensed use of MathWorks products. The process validates your product licenses and ensures that they are used correctly. You must complete the activation process before you can use Polyspace software.

Note If you are using Designated Computer (Individual) licenses, you must activate the license for each Polyspace system individually. However, if you are using Concurrent licenses for multiple Polyspace systems, you do not need to activate each Polyspace system. You activate the license once (for the FLEXnet license server), then provide license files for each Polyspace system.

The easiest way to activate the software is to log in to your MathWorks Account during installation. At the end of the installation process, the Polyspace Software Activation dialog box opens.



Activate PolySpace Software

Activation is a process that verifies licensed use of MathWorks products. This process validates the license and ensures that it is not used on more systems than allowed by the license option you have acquired.

[✓ Activate automatically using the Internet (recommended)]

Activate manually without the Internet

Follow the prompts in the Polyspace Software Activation dialog box to complete the activation process.

If you do not have a MathWorks account, you can create one during the activation process. To create an account, you must have an Activation Key, which identifies the license you want to install and activate.

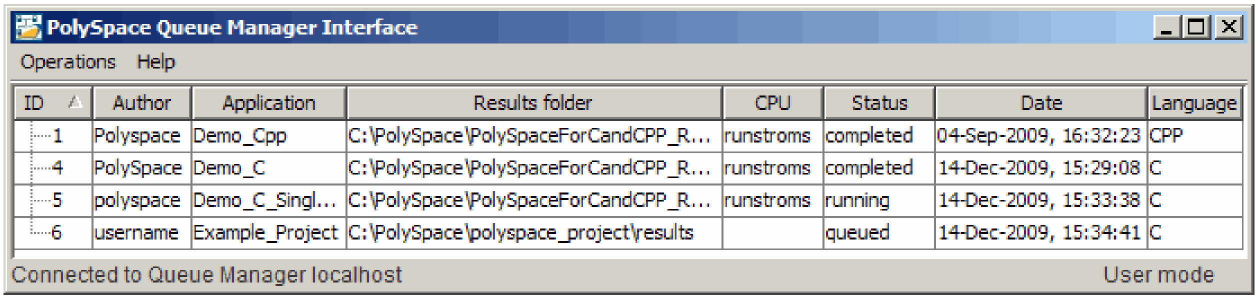
If your Polyspace system is not connected to the internet, you can access the MathWorks License Center on a computer with internet access, activate your license, and download a license file for transfer to your Polyspace system. If you do not have access to a computer with an Internet connection, contact Customer Support.

For more information on how to activate your software, see “Activating Polyspace Software”in the *Polyspace Installation Guide*.

For more information on software activation, including frequently asked questions, refer to the MathWorks Web site:
www.mathworks.com/support/activation/polyspace.html

Queue Manager Interface

The Polyspace Queue Manager Interface (Spooler) is now available on Linux machines, providing a graphical interface for managing verification jobs on the Polyspace server.



ID	Author	Application	Results folder	CPU	Status	Date	Language
1	Polyspace	Demo_Cpp	C:\PolySpace\PolySpaceForCandCPP_R...	runstroms	completed	04-Sep-2009, 16:32:23	CPP
4	PolySpace	Demo_C	C:\PolySpace\PolySpaceForCandCPP_R...	runstroms	completed	14-Dec-2009, 15:29:08	C
5	polyspace	Demo_C_Singl...	C:\PolySpace\PolySpaceForCandCPP_R...	runstroms	running	14-Dec-2009, 15:33:38	C
6	username	Example_Project	C:\PolySpace\polyspace_project\results		queued	14-Dec-2009, 15:34:41	C

Connected to Queue Manager localhost
User mode

For more information, see "Managing Verification Jobs Using Polyspace Queue Manager"in the *Polyspace Products for Ada User's Guide*.

Operating System Support

Added support for the following Linux distributions:

- OpenSuSE 11.1
- Debian 5.x
- Ubuntu 8.04, 8.10, 9.04, and 9.10

For more information, see the *Polyspace Installation Guide*.

Polyspace Model Link SL Product

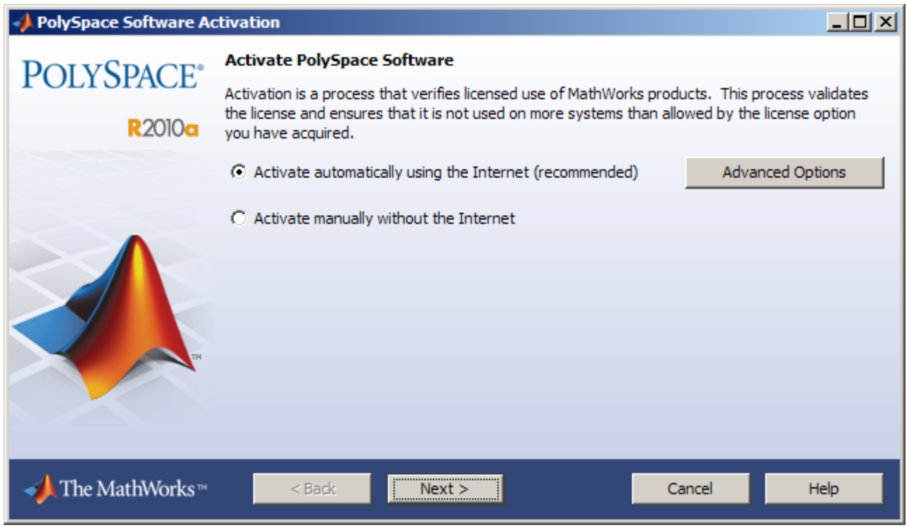
License Activation

Polyspace products now support the MathWorks software activation mechanism.

Activation is a process that verifies licensed use of MathWorks products. The process validates your product licenses and ensures that they are used correctly. You must complete the activation process before you can use Polyspace software.

Note If you are using Designated Computer (Individual) licenses, you must activate the license for each Polyspace system individually. However, if you are using Concurrent licenses for multiple Polyspace systems, you do not need to activate each Polyspace system. You activate the license once (for the FLEXnet license server), then provide license files for each Polyspace system.

The easiest way to activate the software is to log in to your MathWorks Account during installation. At the end of the installation process, the Polyspace Software Activation dialog box opens.



Activate PolySpace Software

Activation is a process that verifies licensed use of MathWorks products. This process validates the license and ensures that it is not used on more systems than allowed by the license option you have acquired.

[✓ Activate automatically using the Internet (recommended)]

Activate manually without the Internet

Follow the prompts in the Polyspace Software Activation dialog box to complete the activation process.

If you do not have a MathWorks account, you can create one during the activation process. To create an account, you must have an Activation Key, which identifies the license you want to install and activate.

If your Polyspace system is not connected to the internet, you can access the MathWorks License Center on a computer with internet access, activate your license, and download a license file for transfer to your Polyspace system. If you do not have access to a computer with an Internet connection, contact Customer Support.

For more information on how to activate your software, see "Activating Polyspace Software"in the *Polyspace Installation Guide*.

For more information on software activation, including frequently asked questions, refer to the MathWorks Web site:
www.mathworks.com/support/activation/polyspace.html

Data Range Specifications for Custom Simulink Data Objects

Polyspace Model Link SL software now accepts every Simulink or `mpt` object containing min and max values.

In previous releases, the software did not create DRS entries for custom Simulink Data Objects, only for `Simulink.Parameter`, `mpt.Parameter`, `Simulink.Signal`, and `mpt.Signal`.

Compatibility Considerations. Verification results may change when compared to previous versions of the software, due to data ranges being applied to additional objects.

Simulink Software Support

Added support for Simulink Version 7.5 (R2010a).

Version 7.1 (R2009b) Polyspace for C/C++ Products

This table summarizes what's new in V7.1 (R2009b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for C/C++ Product" on page 114
- "Polyspace® Server for C/C++ Product" on page 119

Polyspace Client for C/C++ Product

Report Generator

New Report Generator that presents Polyspace results in PDF, HTML, and other output formats.

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

- **Coding Rules Report** – Provides information about compliance with MISRA-C Coding Rules, as well as Polyspace configuration settings for the verification.
- **Developer Report** – Provides information useful to developers, including summary results, detailed lists of red, orange, and gray checks, and Polyspace configuration settings for the verification.
- **Developer with Green Checks Report** – Provides the same content as the Developer Report, but also includes a detailed list of green checks.

- **Quality Report** – Provides information useful to quality engineers, including summary results, statistics about the code, graphs showing distributions of checks per file, and Polyspace configuration settings for the verification.

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

- HTML
- PDF
- RTF
- Microsoft Word
- XML

Note Microsoft Word format is not available on UNIX platforms. RTF format is used instead.

For more information, see "Generating Reports of Verification Results" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Viewer Enhancements

Enhanced Viewer displays results with tooltips containing the values of variables, operands, function parameters, and return values.

You can see range information associated with variables and operators within the source code view.

Note The displayed range information represents a superset of dynamic values, which the software computes using static methods.

If a line of code is all the same color, selecting the line opens an Expanded Source Code window. Place your cursor over the required operator or variable in this window to view range information.

If a line of code contains different colored checks, selecting a check displays the error or warning message along with range information in the selected check view.

For more information, see "Using Range Information in Run-Time Checks Perspective" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Global Data Graphs

New Graphs (similar to concurrent access graphs) available for all global data.

You can display the access sequence for any variable that is read or written in the code. The access graph displays the read and write access for the variable.

For more information, see "Displaying the Access Graph for Variables" in the *Polyspace Products for C User's Guide* or *Polyspace Products for C++ User's Guide*.

Unit-by-unit Verification

New option to create a separate verification job for each source file in the project.

When you run a unit-by-unit verification, each source file is compiled, sent to the Polyspace Server, and verified individually.

The queue manager displays a job for the full verification group, as well as jobs for each unit (using a tree structure).

When verification is complete, you can download and view results for the entire project, or for individual units. When downloading a verification group, all the unit results are downloaded and a summary of the download status for each unit is displayed.

Note Unit by unit verification is available only for server verifications.

For more information, see "Running Verification Unit-by-Unit" in the *Polyspace Products for C Reference* or *Polyspace Products for C++ Reference*.

Changes to Coding Rules Checker Results

- "Compatibility Considerations" on page 117
- "MISRA-C Rule 5.1 Analysis Improved" on page 117
- "MISRA-C Rule 5.2 Analysis Improved" on page 117
- "MISRA-C Rule 5.7 Analysis Improved" on page 117
- "MISRA-C Rule 8.10 Analysis Improved" on page 118
- "MISRA-C Rule 10.1 Analysis Relaxed" on page 118
- "MISRA-C Rule 10.5 Analysis Improved" on page 118
- "MISRA-C Rule 12.7 Analysis Improved" on page 118
- "MISRA-C Rule 15.0 Analysis Improved" on page 118
- "MISRA-C Rule 16.4 Analysis Improved" on page 118

Compatibility Considerations. Due to changes in the coding rules checker, the number of coding rule violations may change when compared to previous versions of the software.

Refer to the following sections for information on the specific changes.

MISRA-C Rule 5.1 Analysis Improved. The coding rules checker now applies MISRA-C rule 5.1 to all identifiers external and internal.

MISRA-C Rule 5.2 Analysis Improved. The coding rules checker now detects violations of MISRA-C Rule 5.2 when the declaration in the outer scope occurs after the declaration in the inner scope.

MISRA-C Rule 5.7 Analysis Improved. The coding rules checker now detects violations of MISRA-C Rule 5.7 in local reused identifiers.

MISRA-C Rule 8.10 Analysis Improved. Only the last declaration takes precedence for `static` or `extern`. The coding rules checker no longer reports violations of MISRA-C Rule 8.10 if the last declaration is static.

MISRA-C Rule 10.1 Analysis Relaxed. The coding rules checker has relaxed enforcement of MISRA-C Rule 10.1 for `x` in `[x]` for any type of expression `x`.

MISRA-C Rule 10.5 Analysis Improved. The coding rules checker now detects violations of MISRA-C Rule 10.5 in expressions with constants.

For example:

```
c = (uint8_t)(ui8 << ( 1U << 2U ) );
```

MISRA-C Rule 12.7 Analysis Improved. The coding rules checker now detects violations of MISRA-C Rule 12.7 in expressions with constants.

For example:

```
~(i=1);
```

MISRA-C Rule 15.0 Analysis Improved. The coding rules checker now detects violations of MISRA-C Rule 15.0 in all statements between switch and first case clause (label, harmless statement).

In addition the coding rules checker now detects jumps and label statements.

MISRA-C Rule 16.4 Analysis Improved. The coding rules checker now keeps the names of the parameters of the first declaration, and reports violations of MISRA-C Rule 16.4 for each occurrence.

Operating System Support

Added support for Windows Server® 2008.

For more information, see the *Polyspace Installation Guide*.

Polyspace Server for C/C++ Product

Operating System Support

Added support for Windows Server 2008.

For more information, see the *Polyspace Installation Guide*.

Version 5.4 (R2009b) Polyspace for Ada and Model Link Products

This table summarizes what's new in V5.4 (R2009b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports Polyspace Model Link SL Bug Reports Polyspace Model Link TL Bug Reports Polyspace UML Link RH Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for Ada Product" on page 120
- "Polyspace® Server for Ada Product" on page 127
- "Polyspace Model Link SL Product" on page 127

Polyspace Client for Ada Product

Report Generator

New Report Generator that presents Polyspace results in PDF, HTML, and other output formats.

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

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

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

- HTML
- PDF
- RTF
- Microsoft Word
- XML

Note Microsoft Word format is not available on UNIX platforms. RTF format is used instead.

For more information, see "Generating Reports of Verification Results" in the *Polyspace Products for Ada User's Guide* .

Main Generator Enhancements

Enhanced main generator that considers the scope of a procedure and variable, improving error detection at the package level.

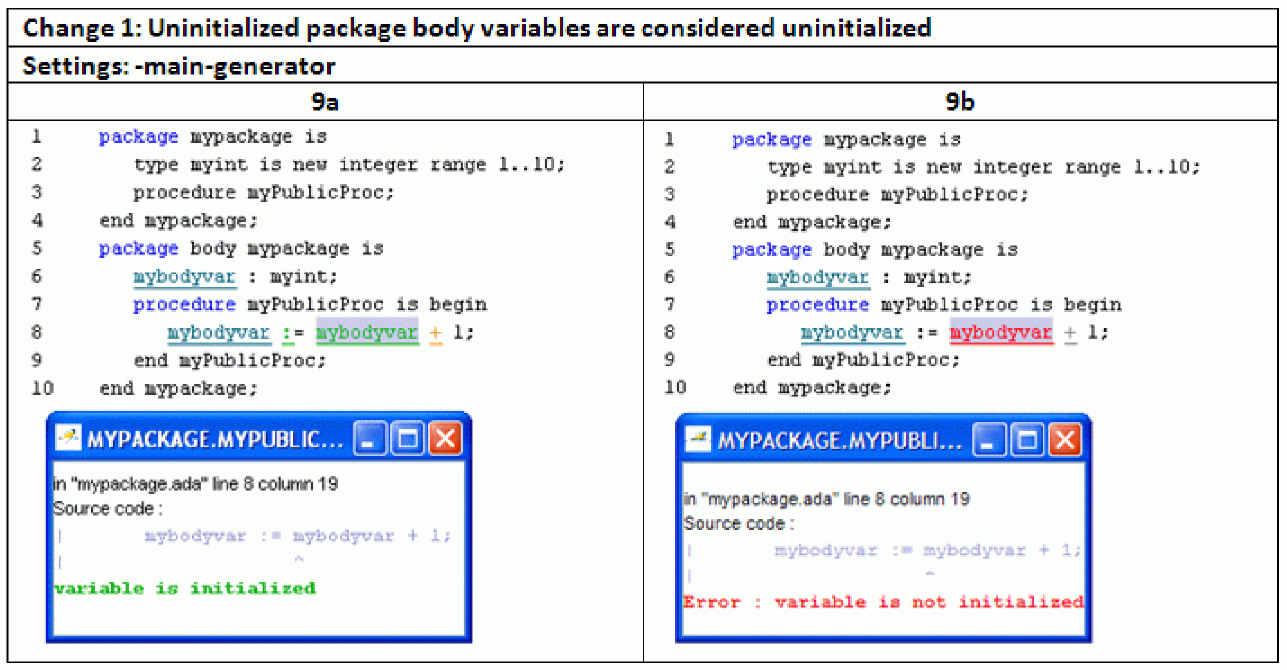
This change may affect your results compared with previous releases, and how you interpret the new results. Specific changes include:

- Uninitialized package body variables are considered uninitialized
- Uncalled package-scope procedures/functions are considered unreachable
- Functions/procedures declared at the spec level are called only once
- Uninitialized spec level variables are considered possibly uninitialized

For more information on the main generator, see "Main Generator Overview" in the *Polyspace Products for Ada User's Guide* .

Uninitialized Package Body Variables are Considered Uninitialized . Uninitialized variables that are declared only in the package body are now considered uninitialized, and generate red `NIV` checks.

Previously, these variables were considered initialized (green `NIV`) with full-range values. This behaviour made interpretation of results easier and allowed verification to continue. However, the software now considers these variables uninitialized (red `NIV`) and stops verification at this point. This new behaviour is more accurate with respect to the actual initialization state of the variables. You must correct the code before verification can continue. Alternatively, you can use the option `-continue-with-all-niv`.



Change 1: Uninitialized package body variables are considered uninitialized

Settings: -main-generator

9a

```
1     package mypackage is
2        type myint is new integer range 1..10;
3        procedure myPublicProc;
4     end mypackage;
5     package body mypackage is
6        mybodyvar : myint;
7        procedure myPublicProc is begin
8           mybodyvar := mybodyvar + 1;
9        end myPublicProc;
10    end mypackage;
```

MYPACKAGE.MYPUBLIC...

```
in "mypackage.ada" line 8 column 19
Source code :
|        mybodyvar := mybodyvar + 1;
|                     ^
variable is initialized
```

9b

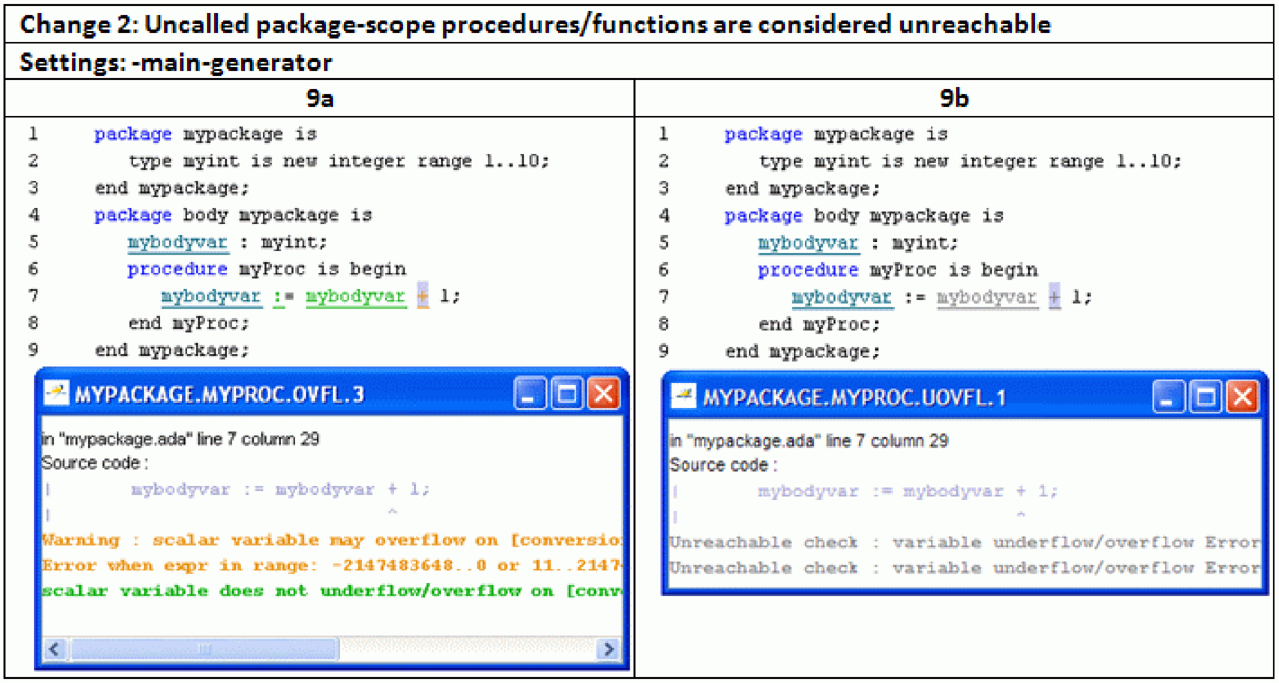
```
1     package mypackage is
2        type myint is new integer range 1..10;
3        procedure myPublicProc;
4     end mypackage;
5     package body mypackage is
6        mybodyvar : myint;
7        procedure myPublicProc is begin
8           mybodyvar := mybodyvar + 1;
9        end myPublicProc;
10    end mypackage;
```

MYPACKAGE.MYPUBLI...

```
in "mypackage.ada" line 8 column 19
Source code :
|        mybodyvar := mybodyvar + 1;
|                     ^
Error : variable is not initialized
```

Uncalled Package-scope Procedures/Functions are Considered Unreachable . Procedures and functions that are declared in the package but not called by code within the package body provided for the verification will now be considered unreachable (gray).

Previously, all procedures and functions in a package were considered for verification and subsequently colored. The argument for this behavior was that these functions and procedures could be called by code inside the package that had not been provided for the verification. Now, the software considers this code unreachable (gray) unless there is a path of execution that leads to it.



Change 2: Uncalled package-scope procedures/functions are considered unreachable

Settings: -main-generator

9a

```
package mypackage is
   type myint is new integer range 1..10;
end mypackage;
package body mypackage is
   mybodyvar : myint;
   procedure myProc is begin
      mybodyvar := mybodyvar + 1;
   end myProc;
end mypackage;
```

MYPACKAGE.MYPROC.OVFL.3

```
in "mypackage.ada" line 7 column 29
Source code :
|        mybodyvar := mybodyvar + 1;
|                                ^
Warning : scalar variable may overflow on [conversio
Error when expr in range: -2147483648..0 or 11..2147
scalar variable does not underflow/overflow on [conv
```

9b

```
package mypackage is
   type myint is new integer range 1..10;
end mypackage;
package body mypackage is
   mybodyvar : myint;
   procedure myProc is begin
      mybodyvar := mybodyvar + 1;
   end myProc;
end mypackage;
```

MYPACKAGE.MYPROC.UOVFL.1

```
in "mypackage.ada" line 7 column 29
Source code :
|        mybodyvar := mybodyvar + 1;
|                                ^
Unreachable check : variable underflow/overflow Error
Unreachable check : variable underflow/overflow Error
```

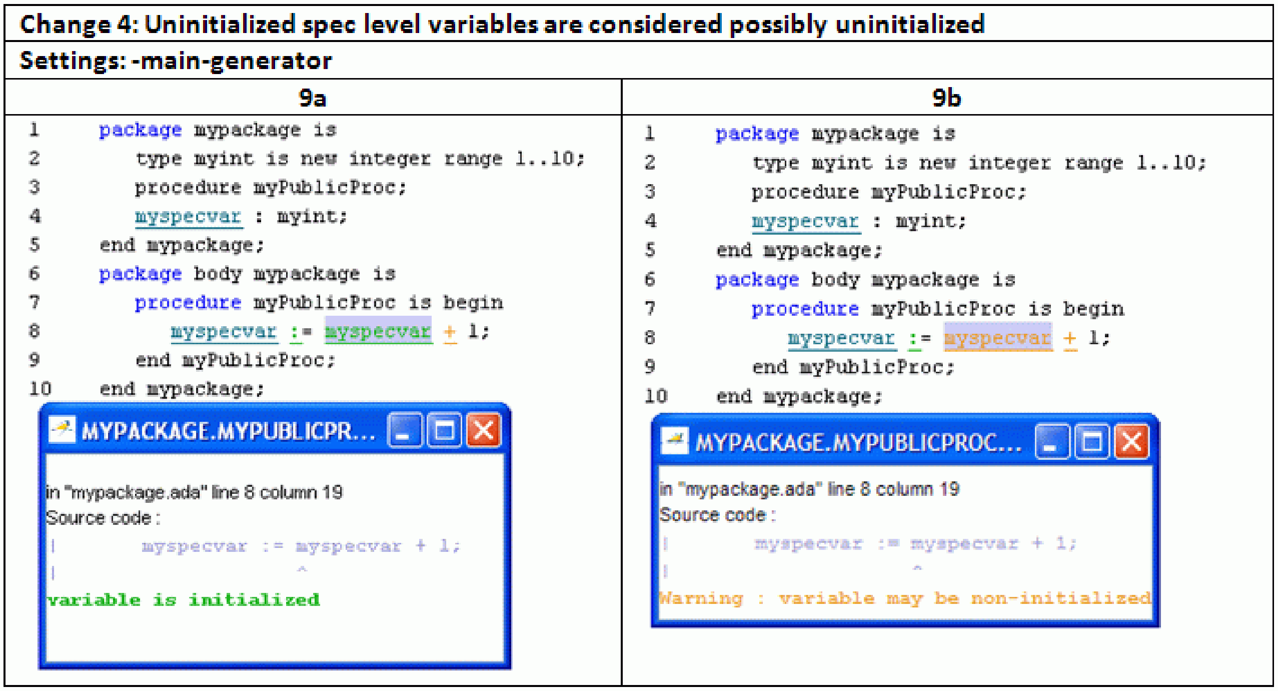
Functions/Procedures Declared at the Spec Level are Called Only Once. Functions or procedures declared at the specification level are called only once.

Note This behavior changed in Release 2010a. In R2010a or later, the main generator can call a function several times.

Uninitialized Spec Level Variables are Considered Possibly Uninitialized. Uninitialized variables declared in a package specification will now be considered possibly uninitialized (orange `NIV`). Previously, these variables were considered initialized (green `NIV`) with full-range values.

The software now considers uninitialized variables that are declared only in the package *body* as uninitialized (red `NIV`). However, for uninitialized variables declared in a package *specification*, it is possible that packages that use these variables may initialize these variables. The software now recognizes this possibility and generates orange `NIV` checks for uninitialized variables declared in a package specification.

This behavior is not changed if you use options `-init-stubbing-vars-random` or `-init-stubbing-vars-zero-or-random` to initialize uninitialized variables. Specification-level variables will still be considered possibly uninitialized (orange `NIV`), because the packages that use these variables can alter the variables, even to the extent of uninitializing the variables.



Change 4: Uninitialized spec level variables are considered possibly uninitialized

Settings: -main-generator

9a

```
package mypackage is
   type myint is new integer range 1..10;
   procedure myPublicProc;
   myspecvar : myint;
end mypackage;
package body mypackage is
   procedure myPublicProc is begin
      myspecvar := myspecvar + 1;
   end myPublicProc;
end mypackage;
```

MYPACKAGE.MYPUBLICPR...

```
in "mypackage.ada" line 8 column 19
Source code :
|        myspecvar := myspecvar + 1;
|                     ^
variable is initialized
```

9b

```
package mypackage is
   type myint is new integer range 1..10;
   procedure myPublicProc;
   myspecvar : myint;
end mypackage;
package body mypackage is
   procedure myPublicProc is begin
      myspecvar := myspecvar + 1;
   end myPublicProc;
end mypackage;
```

MYPACKAGE.MYPUBLICPROC...

```
in "mypackage.ada" line 8 column 19
Source code :
|        myspecvar := myspecvar + 1;
|                     ^
Warning : variable may be non-initialized
```

Compatibility Considerations. Changes to the main generator may result in differences in your verification results, when compared with earlier versions of the software. If you verified your code with previous versions of the software (for example, R2009a), be aware of these changes, how they affect your colored results and the way you interpret the results.

Global Data Graphs

New Graphs (similar to concurrent access graphs) available for all global data.

You can display the access sequence for any variable that is read or written in the code. The access graph displays the read and write access for the variable.

For more information, see “Displaying the Access Graph for Variables” in the *Polyspace Products for Ada User’s Guide* .

Unit-by-unit Verification

New option to create a separate verification job for each source file in the project.

When you run a unit-by-unit verification, each source file is compiled, sent to the Polyspace Server, and verified individually.

The queue manager displays a job for the full verification group, as well as jobs for each unit (using a tree structure).

When verification is complete, you can download and view results for the entire project, or for individual units. When downloading a verification group, all the unit results are downloaded and a summary of the download status for each unit is displayed.

Note Unit by unit verification is available only for server verifications.

For more information, see “Running Verification Unit-by-Unit” in the *Polyspace Products for Ada Reference*.

Operating System Support

Added support for Windows Server 2008.

For more information, see the *Polyspace Installation Guide*.

Polyspace Server for Ada Product

Operating System Support

Added support for Windows Server 2008.

For more information, see the *Polyspace Installation Guide*.

Polyspace Model Link SL Product

Simulink Software Support

Added support for Simulink Version 7.4 (R2009b).

Version 7.0 (R2009a) Polyspace for C/C++ Products

This table summarizes what's new in V7.0 (R2009a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for C/C++ Product" on page 128
- "Polyspace® Server for C/C++ Product" on page 133

Polyspace Client for C/C++ Product

JSF++ Support

Enhanced JSF C++ checker supports all checkable Joint Strike Fighter Air Vehicle C++ coding standards (JSF++:2005).

Polyspace software can now check all possible C++ programming rules defined by Lockheed Martin® for the JSF program. These coding standards are designed to improve the robustness of C++ code, and improve maintainability.

For more information, see "Checking Coding Rules", in the *Polyspace Products for C++ User's Guide*.

Back to Source Link

New "back-to-source" link in the Polyspace launcher associates compile errors, MISRA-C violations, and JSF++ violations reported in the logs directly to the source file.

For more information, see “Viewing MISRA C Checker Results”in the *Polyspace Products for C User’s Guide* or “Examining Rule Violations”, in the *Polyspace Products for C++ User’s Guide.*

Eclipse Integration

New Polyspace integration with the Eclipse IDE, Version 3.3.

The Polyspace Client for C/C++ product can be integrated with the Eclipse Integrated Development Environment through the Polyspace C/C++ plug-in for Eclipse IDE.

This plug-in provides Polyspace source code verification and bug detection functionality for source code developed within Eclipse IDE. Features include the following:

- A contextual menu that allows you to launch a verification of one or more files.
- Views in the Eclipse editor that allow you to set verification parameters and monitor verification progress.

For more information, see “Using Polyspace Software in the Eclipse IDE” in the *Polyspace Products for C User’s Guide*or
“Using Polyspace Software in the Eclipse IDE”in the *Polyspace Products for C++ User’s Guide.*

Performance Improvements for Multi-Core Systems

Enhanced performance on multi-core architecture platforms, improving the speed of Polyspace code verification.

The time required to perform an average code verification has been reduced. On multi-core systems, you can now select the number of processes that can run simultaneously, further improving performance.

For more information, see “Number of processes for multiple CPU core systems (`-max-processes`)” in the *Polyspace Products for C Reference*or *Polyspace Products for C++ Reference.*

Architecture Improvements

Several changes have been made to the Polyspace architecture to improve overall performance, as well as the precision of verification results.

During each verification phase (pass), the software now only analyzes those procedures that need to be analyzed. This means that starting with PASS1, if the verification cannot be more precise than that already completed in a previous pass, the procedure is not analyzed again. This improves the overall performance of the verification. It also means that some passes will finish more quickly than others, and some passes could be completely empty. This is normal behavior.

In addition, these architecture improvements result in the following changes:

- The `quick` precision option is now obsolete, and has been removed. `quick` mode has been replaced with verification PASS0. PASS0 takes somewhat longer to run, but the results are more complete. The limitations of `quick` mode, (no NTL or NTC checks, no float checks, no variable dictionary) no longer apply. Unlike `quick` mode, PASS0 also provides full navigation in the Viewer.
- The `voa` option is now obsolete, and has been removed. Value On Assignment checks are now provided by default. In general, this means that Polyspace results now contain many more VOA checks. For C applications, all possible VOA are given.
- The UOVFL (Float Underflows and Overflows) check no longer exists. Float underflows and overflows are now reported as two separate checks. This is similar to the way integers are handled.

  Note Since the single UOVFL check has been replaced by two checks, the total number of checks reported by Polyspace on a given file may be different in this release than with previous versions of the software.

- Messages have been improved for float arithmetic checks, making them similar to the messages for integers. For example, NIV checks on float variables now contain the type size (32 or 64).

- For IPT (Inspection Point) checks, there is now one check for each variable. Previously there was a single IPT check (on the keyword) for multiple variables.
- The log file has several additions, including the names of each PASS, the verification phases, and additional messages.

Compatibility Considerations. The verification results provided by Polyspace software may be different in R2009a than with previous releases of the software. Verification results are more precise, and the total number of checks reported on a given source file may be different. In general, the software now reports more checks, due to increased VOA checks, changes to the IPT check, and the single float UOVFL check being replaced by two checks (UNFL and OVFL).

In addition, due to the float UOVFL check being split into two checks, the selectivity (number of proven checks red+green+gray / number of total checks) of a verification may change significantly for applications using many float variables. For example, an application that had 10 orange UOVFL checks with a previous release, could now have up to 20 orange UNFL and OVFL checks on the same float variables. Although this appears to be a decrease in precision, the verification itself is not less precise.

Mathematical Functions Included in Stubs

Mathematical functions are now included in the standard stubs. This means:

- An IRV (Initialized Return Value) check appears on the math function call.
- The POW check no longer appears in the Viewer.
- Math functions appear in the call graph.
- The modeling of mathematical functions is visible through the stub body, instead of being handled internally.
- By default, math functions are launched with the option `-context-sensitivity` , allowing them to distinguish their calling sites.

In addition, you can provide your own math functions instead of using the standard stub provided by Polyspace software. This allows the software to verify the body of the math function, instead of using a stub for the math function.

For example, in C90, the mathematical function `fabs()` has the prototype:

```
double fabs(double) ;
```

However, on a 16-bit target, the function may have the prototype:

```
float fabs(float);
```

In this case, you would want to verify your own `fabs()` function.

To provide your own math function:

1 Create source code for the function. For example:

```
float  fabs (float var)
{
  if (var >= 0.0f)
    return var;
  return -var;
}
```

2 Provide the function to your verification using the `D` compiler flag. For example:

```
polyspace-c -D  __polyspace_no_fabs
```

Note There is a compiler flag for each standard ANSI C90 mathematical function. A complete list of flags is located in the file: `%POLYSPACE_C%\Verifier\cinclude\__polyspace__stdstubs.c`.

Compatibility Considerations. Since the POW check no longer appears in the Viewer, verification results may be different in R2009a than with previous releases of the software.

Character Encoding Options

New character encoding option allows you to view source files created on an operating system that uses different character encoding than your current system.

You specify the character encoding used by the operating system on which the source file was created using the **Character encoding** tab in the Preferences dialog box of the Polyspace Viewer.

For more information, see "Setting Character Encoding Preferences"in the *Polyspace Products for C User's Guide*or *Polyspace Products for C++ User's Guide*.

Automatic Orange Tester

The Automatic Orange Tester (for C), dynamically stresses unproven code (orange checks) to help you identify run-time errors.

For more information, see "Automatically Testing Orange Code" in the *Polyspace Products for C User's Guide*.

Compatibility Considerations. If you open verification results created with an older version of the product in the Automatic Orange Tester, you may get a compilation error. The version of the product used to create the instrumented source code must be the same as the one used for analysis in the Automatic Orange Tester.

To avoid this problem, re-launch the code verification with the current version of the product.

Operating System Support

Added support for Windows Server 2003, Windows Vista™, and Red Hat Enterprise Linux Workstation v.5.

For more information, see the *Polyspace Installation Guide*.

Polyspace Server for C/C++ Product

Performance Improvements for Multi-Core Systems

Enhanced performance on multi-core architecture platforms, improving the speed of Polyspace code verification.

The time required to perform an average code verification has been reduced. On multi-core systems, you can now select the number of processes that can run simultaneously, further improving performance.

For more information, see "Number of processes for multiple CPU core systems (`-max-processes`)" in the *Polyspace Products for C Reference*or *Polyspace Products for C++ Reference*.

Architecture Improvements

Several changes have been made to the Polyspace architecture to improve overall performance, as well as the precision of verification results.

During each verification phase (pass), the software now only analyzes those procedures that need to be analyzed. This means that starting with PASS1, if the verification cannot be more precise than that already completed in a previous pass, the procedure is not analyzed again. This improves the overall performance of the verification. It also means that some passes will finish more quickly than others, and some passes could be completely empty. This is normal behavior.

In addition, these architecture improvements result in the following changes:

- The `quick` precision option is now obsolete, and has been removed. `quick` mode has been replaced with verification PASS0. PASS0 takes somewhat longer to run, but the results are more complete. The limitations of `quick` mode, (no NTL or NTC checks, no float checks, no variable dictionary) no longer apply. Unlike `quick` mode, PASS0 also provides full navigation in the Viewer.
- The `voa` option is now obsolete, and has been removed. Value On Assignment checks are now provided by default. In C, all possible VOA are given.
- The UOVFL (Float Underflows and Overflows) check no longer exists. Float underflows and overflows are now reported as two separate checks. This is similar to the way integers are handled.

Note Since the single UOVFL check has been replaced by two checks, the total number of checks reported by Polyspace on a given file may be different in this release than with previous versions of the software.

- Messages have been improved for float arithmetic checks, making them similar to the messages for integers. For example, NIV checks on float variables now contain the type size (32 or 64).
- For IPT (Inspection Point) checks, there is now one check for each variable. Previously there was a single IPT check (on the keyword) for multiple variables.
- The log file has several additions, including the names of each PASS, the verification phases, and additional messages.

Compatibility Considerations. The verification results provided by Polyspace software may be different in R2009a than with previous releases of the software. Verification results are more precise, and the total number of checks reported on a given source file may be different. In general, the software now reports more checks, due to increased VOA checks, changes to the IPT check, and the single float UOVFL check being replaced by two checks (UNFL and OVFL).

In addition, due to the float UOVFL check being split into two checks, the selectivity (number of proven checks red+green+gray / number of total checks) of a verification may change significantly for applications using many float variables. For example, an application that had 10 orange UOVFL checks with a previous release, could now have up to 20 orange UNFL and OVFL checks on the same float variables. Although this appears to be a decrease in precision, the verification itself is not less precise.

Operating System Support

Added support for Windows Server 2003, Windows Vista, and Red Hat Enterprise Linux Workstation v.5.

For more information, see the *Polyspace Installation Guide*.

Version 5.3 (R2009a) Polyspace for Ada and Model Link Products

This table summarizes what's new in V5.3 (R2009a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	No	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports Polyspace Model Link SL Bug Reports Polyspace Model Link TL Bug Reports Polyspace UML Link RH Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for Ada Product" on page 136
- "Polyspace® Server for Ada Product" on page 137
- "Polyspace Model Link SL Product" on page 137
- "Polyspace UML Link RH Product" on page 138

Polyspace Client for Ada Product

Character Encoding Options

New character encoding option allows you to view source files created on an operating system that uses different character encoding than your current system.

You specify the character encoding used by the operating system on which the source file was created using the **Character encoding** tab in the Preferences dialog box of the Polyspace Viewer.

For more information, see “Setting Character Encoding Preferences” in the *Polyspace Products for Ada User’s Guide*.

Operating System Support

Added support for Windows Server 2003, Windows Vista, and Red Hat Enterprise Linux Workstation v.5.

For more information, see the *Polyspace Installation Guide*.

Polyspace Server for Ada Product

Operating System Support

Added support for Windows Server 2003, Windows Vista, and Red Hat Enterprise Linux Workstation v.5.

For more information, see the *Polyspace Installation Guide*.

Polyspace Model Link SL Product

Polyspace Menu Option in Simulink

New option in the Simulink Tools menu to launch Polyspace software directly from Simulink.

For more information, see “Starting the Polyspace Verification”in the *Polyspace Model Link Products User’s Guide*.

Manual Selection of Data Range Specifications (DRS) File

You can now manually select a Data Range Specification (DRS) file within Simulink, instead of accepting the default DRS file.

For more information, see “Data Range Specification”in the *Polyspace Model Link Products User’s Guide*.

Simulink Software Support

Added support for Simulink Version 7.3 (R2009a).

Polyspace UML Link RH Product

Rhapsody Support

Added support for Telelogic Rhapsody Version 7.2 and 7.3.

Version 6.0 (R2008b) Polyspace for C/C++ Products

This table summarizes what's new in V6.0 (R2008b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	No	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for C/C++ Product" on page 139
- "Polyspace® Server for C/C++ Product" on page 140

Polyspace Client for C/C++ Product

Automatic Orange Tester

Automatic Orange Tester (for C), dynamically stresses unproven code (orange checks) to identify run-time errors, and provides information to help you identify the cause of these errors.

The Automatic Orange Tester complements the results review in the Viewer module of Polyspace Client for C/C++ by automatically creating test cases for all input variables in orange code, and then dynamically testing the code to find actual runtime errors. The Automatic Orange Tester also provides detailed information on why each test-case failed. You can use this information to quickly identify the cause of the error, and determine if there is an actual bug in the code.

For more information, see "Automatically Testing Orange Code" in the *Polyspace Products for C User's Guide*.

JSF++ Support

Support for a subset of the Joint Strike Fighter Air Vehicle C++ coding standards (JSF++:2005).

Polyspace software can now check 120 of the C++ programming rules defined by Lockheed Martin for the JSF program. These coding standards are designed to improve the robustness of C++ code, and improve maintainability.

For more information, see “Checking Coding Rules”, in the *Polyspace Products for C++ User’s Guide*.

Operating System Support

Added support for 64–bit Linux.

For more information, see the *Polyspace Installation Guide*.

Polyspace Server for C/C++ Product

Operating System Support

Added support for 64–bit Linux.

For more information, see the *Polyspace Installation Guide*.

Version 5.2 (R2008b) Polyspace for Ada and Model Link Products

This table summarizes what's new in V5.2 (R2008b):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	No	Includes fixes: Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports Polyspace Model Link SL Bug Reports

New features and changes introduced in this version are organized by product:

- "Polyspace® Client for Ada Product" on page 141
- "Polyspace® Server for Ada Product" on page 141
- "Polyspace Model Link SL Product" on page 142
- "Polyspace Model Link TL Product" on page 142
- "Polyspace UML Link RH Product" on page 143

Polyspace Client for Ada Product

Operating System Support

Added support for 64–bit Linux.

For more information, see the *Polyspace Installation Guide*.

Polyspace Server for Ada Product

Operating System Support

Added support for 64–bit Linux.

For more information, see the *Polyspace Installation Guide*.

Polyspace Model Link SL Product

Model Reference Support

Added support for Simulink Model Reference.

Polyspace Model Link SL software now automatically detects model references in Simulink models, allowing you to quickly track any verification issues back to the original model.

For more information, see the *Polyspace Model Link Products User's Guide*.

Stateflow Chart Support

Added support for Stateflow® Charts within Simulink models.

Polyspace Model Link SL software now supports Stateflow Charts within Simulink models, allowing you to quickly track any verification issues back to the original Stateflow chart. In addition, any Stateflow comments are now highlighted in the Polyspace source code view.

For more information, see the *Polyspace Model Link Products User's Guide*.

Simulink Software Support

Added support for Simulink Version 7.2 (R2008b).

Operating System Support

Added support for 64–bit Linux.

For more information, see the *Polyspace Installation Guide*.

Polyspace Model Link TL Product

Operating System Support

Added support for 64–bit Linux.

For more information, see the *Polyspace Installation Guide*.

Polyspace UML Link RH Product

Ada Language Support

Added support for Ada language in Rhapsody software.

For more information, see the *Polyspace UML Link™ RH User's Guide*.

Operating System Support

Added support for 64–bit Linux.

For more information, see the *Polyspace Installation Guide*.

Version 5.1 (R2008a) Polyspace Software

This table summarizes what's new in V5.1 (R2008a):

New Features and Changes	Version Compatibility Considerations	Fixed Bugs and Known Problems
Yes Details below	Yes—Details labeled as **Compatibility Considerations**, below. See also Summary.	Includes fixes: Polyspace Client for C/C++ Bug Reports Polyspace Server for C/C++ Bug Reports Polyspace Client for Ada Bug Reports Polyspace Server for Ada Bug Reports Polyspace Model Link SL Bug Reports

New features and changes introduced in this version are organized by product:

Polyspace Client for Ada Product

Removed Cygwin Software Dependency for Windows Platforms

Previous versions of Polyspace products used Cygwin™ emulation to run UNIX® commands on Windows systems.

In version 5.1, the Cygwin software dependency has been removed. Removing Cygwin simplifies the Polyspace product installation process while improving the performance and robustness of the Polyspace Verification process.

Compatibility Considerations. Due to the Cygwin changes, Polyspace Client for Ada Version 5.1 is not compatible with previous versions of Polyspace products on Windows platforms. To avoid compatibility problems on Windows platforms, you must upgrade all your Polyspace client and server products at the same time.

If your Polyspace server is running on a Windows platform, the binary files used for batch commands in previous releases will not work without Cygwin software installed. In version 5.1, the software provides new .exe files for these batch commands. However, these files are now located in a different location.

Commands	Previous Location	New Location
Standard	`*PolyspaceInstallDir*\verifier\bin\`	`*PolyspaceInstallDir*\verifier\wbin\`
Remote Launcher	`*Polyspace_Common*\RemoteLauncher\bin\`	`*Polyspace_Common*\RemoteLauncher\wbin\`
Viewer	`*Polyspace_Common*\Viewer\bin\`	`*Polyspace_Common*\Viewer\wbin\`

If you wrote scripts using batch commands in previous releases, you must modify the scripts to use the new commands.

In addition, if you used Cygwin shell scripts for postprocessing or target compilation, those scripts will no longer run on version 5.1. To support scripting, the Polyspace software now includes Perl. You can access Perl in:

```
PolyspaceInstallDir\verifier\tools\perl\win32\bin\perl.exe
```

Enhanced Installer

Version 5.1 includes an enhanced and simplified installer for all Polyspace products. The installation process is now faster and easier to complete than in previous releases.

For more information, see the *Polyspace Installation Guide*.

Viewer Improvements

Enhanced exploring capability in the viewer to provide more focused information.

Unnecessary information has been eliminated from the Procedural Entities (RTE) View and Call Tree View to improve usability.

Enhanced Compilation Checks

Enhanced compilation checks to stop verification only when a pointer to a task is initiated or used, rather than when it is declared.

One-Click Enhancements

Enhanced Polyspace-In-One-Click options, to allow switching between multiple projects using a browse history.

Operating System Support

Added support for the following operating systems:

- Solaris 2.10
- Windows XP x64 (32-bit mode)

For more information, see the *Polyspace Installation Guide*.

Polyspace Server for Ada Product

Removed Cygwin Software Dependency for Windows Platforms

Previous versions of Polyspace products used Cygwin emulation to run UNIX commands on Windows systems.

In version 5.1, the Cygwin software dependency has been removed. Removing Cygwin simplifies the Polyspace product installation process while improving the performance and robustness of the Polyspace Verification process.

Compatibility Considerations. Due to the Cygwin changes, Polyspace® Server™ for Ada Version 5.1 is not compatible with previous versions of Polyspace products on Windows platforms. To avoid compatibility problems on Windows platforms, you must upgrade all your Polyspace client and server products at the same time.

If your Polyspace server is running on a Windows platform, the binary files used for batch commands in previous releases will not work without Cygwin software installed. In version 5.1, the software provides new .exe files for these batch commands. However, these files are now located in a different location.

Commands	Previous Location	New Location
Standard	`PolyspaceInstallDir\verifier\bin\`	`PolyspaceInstallDir\verifier\wbin\`
Remote Launcher	`Polyspace_Common\RemoteLauncher\bin\`	`Polyspace_Common\RemoteLauncher\wbin\`
Viewer	`Polyspace_Common\Viewer\bin\`	`Polyspace_Common\Viewer\wbin\`

If you wrote scripts using batch commands in previous releases, you must modify the scripts to use the new commands.

In addition, if you used Cygwin shell scripts for postprocessing or target compilation, those scripts will no longer run on version 5.1. To support scripting, the Polyspace software now includes Perl. You can access Perl in:

```
PolyspaceInstallDir\verifier\tools\perl\win32\bin\perl.exe
```

Enhanced Installer

Version 5.1 includes an enhanced and simplified installer for all Polyspace products. The installation process is now faster and easier to complete than in previous releases.

For more information, see the *Polyspace Installation Guide*.

Operating System Support

Added support for the following operating systems:

- Solaris 2.10
- Windows XP x64 (32-bit mode)

For more information, see the *Polyspace Installation Guide*.

Polyspace Client for C/C++ Product

Removed Cygwin Software Dependency for Windows Platforms

Previous versions of Polyspace products used Cygwin emulation to run UNIX commands on Windows systems.

In version 5.1, the Cygwin software dependency has been removed. Removing Cygwin simplifies the Polyspace product installation process while improving the performance and robustness of the Polyspace Verification process.

Compatibility Considerations. Due to the Cygwin changes, Polyspace Client for C/C++ Version 5.1 is not compatible with previous versions of Polyspace products on Windows platforms. To avoid compatibility problems on Windows platforms, you must upgrade all your Polyspace client and server products at the same time.

If your Polyspace server is running on a Windows platform, the binary files used for batch commands in previous releases will not work without Cygwin software installed. In version 5.1, the software provides new .exe files for these batch commands. However, these files are now located in a different location.

Commands	Previous Location	New Location
Standard	`*PolyspaceInstallDir*\verifier\bin\`	`*PolyspaceInstallDir*\verifier\wbin\`
Remote Launcher	*Polyspace_Common*\ `RemoteLauncher\bin\`	*Polyspace_Common*\ `RemoteLauncher\wbin\`
Viewer	*Polyspace_Common*\ `Viewer\bin\`	*Polyspace_Common*\ `Viewer\wbin\`

If you wrote scripts using batch commands in previous releases, you must modify the scripts to use the new commands.

In addition, if you used Cygwin shell scripts for postprocessing or target compilation, those scripts will no longer run on version 5.1. To support scripting, the Polyspace software now includes Perl. You can access Perl in:

```
PolyspaceInstallDir\verifier\tools\perl\win32\bin\perl.exe
```

Enhanced Installer

Version 5.1 includes an enhanced and simplified installer for all Polyspace products. The installation process is now faster and easier to complete than in previous releases.

For more information, see the *Polyspace Installation Guide*.

Viewer Improvements

Enhanced exploring capability in the viewer to provide more precise locations for C++ checks.

The source code view of the Polyspace viewer now displays the location of C++ checks more accurately.

One-Click Enhancements

Enhanced Polyspace-In-One-Click options, to allow switching between multiple projects using a browse history.

For more information, see "Day to Day Use " in the *Polyspace Products for C User's Guide*.

Generic Target Option for C++

New Generic Target option for C++, to allow custom target processors. The Generic Target option for C++ is similar to the previous Generic Target for C.

For more information, see "Defining Generic Targets" in the *Polyspace Products for C++ User's Guide*.

Class Analyzer Enhancements for C++

Enhanced class analyzer now calls all private constructors and destructors.

Previously, the sources analyzed were generally non-inherited public or protected methods of the class. In version 5.1, the functions that are analyzed include all non-inherited constructors and destructors, and all non-inherited public or protected methods of the class.

For more information, see "Polyspace Class Analyzer" in the *Polyspace Products for C++ User's Guide*.

GNU Compiler Support for C++

New support for the GNU® compiler (GCC 3.4) for C++.

The new GNU dialect option supports variable length arrays, anonymous structures, and other constructions allowed by GCC.

For more information, see "Dialect Issues" in the *Polyspace Products for C++ User's Guide*.

Polyspace C++ Add-in for Visual Studio

Simplified user interface for Polyspace C++ add-in for Microsoft® Visual Studio®.

The Polyspace Browser tab has been eliminated from the Visual Studio window. To perform an analysis of a file in Visual Studio, you now simply right-click on the file and select **Start Polyspace**.

For more information, see "Using Polyspace Software in Visual Studio" in the *Polyspace Products for C++ User's Guide*.

Operating System Support

Added support for the following operating systems:

- Solaris 2.10
- Windows XP x64 (32-bit mode)

For more information, see the *Polyspace Installation Guide*.

Polyspace Server for C/C++ Product

Removed Cygwin Software Dependency for Windows Platforms

Previous versions of Polyspace products used Cygwin emulation to run UNIX commands on Windows systems.

In version 5.1, the Cygwin software dependency has been removed. Removing Cygwin simplifies the Polyspace product installation process while improving the performance and robustness of the Polyspace Verification process.

Compatibility Considerations. Due to the Cygwin changes, Polyspace Server for C/C++ Version 5.1 is not compatible with previous versions of Polyspace products on Windows platforms. To avoid compatibility problems on Windows platforms, you must upgrade all your Polyspace client and server products at the same time.

If your Polyspace server is running on a Windows platform, the binary files used for batch commands in previous releases will not work without Cygwin software installed. In version 5.1, the software provides new .exe files for these batch commands. However, these files are now located in a different location.

Commands	Previous Location	New Location
Standard	`*PolyspaceInstallDir*\verifier\bin\`	`*PolyspaceInstallDir*\verifier\wbin\`
Remote Launcher	`*Polyspace_Common*\RemoteLauncher\bin\`	`*Polyspace_Common*\RemoteLauncher\wbin\`
Viewer	`*Polyspace_Common*\Viewer\bin\`	`*Polyspace_Common*\Viewer\wbin\`

If you wrote scripts using batch commands in previous releases, you must modify the scripts to use the new commands.

In addition, if you used Cygwin shell scripts for postprocessing or target compilation, those scripts will no longer run on version 5.1. To support scripting, the Polyspace software now includes Perl. You can access Perl in:

```
PolyspaceInstallDir\verifier\tools\perl\win32\bin\perl.exe
```

Enhanced Installer

Version 5.1 includes an enhanced and simplified installer for all Polyspace products. The installation process is now faster and easier to complete than in previous releases.

For more information, see the *Polyspace Installation Guide*.

GNU Compiler Support for C++

New support for the GNU compiler (GCC 3.4) for C++.

The new GNU dialect option supports variable length arrays, anonymous structures, and other constructions allowed by GCC.

For more information, see "Dialect Issues" in the *Polyspace Products for C++ User's Guide*.

Operating System Support

Added support for the following operating systems:

- Solaris 2.10
- Windows XP x64 (32-bit mode)

For more information, see the *Polyspace Installation Guide*.

Polyspace Model Link SL Product

Enhanced Installer

Version 5.1 includes an enhanced and simplified installer for all Polyspace products. The installation process is now faster and easier to complete than in previous releases.

For more information, see the *Polyspace Installation Guide*.

Simulink Software Support

Added support for Simulink Version 7.1 (R2008a).

Operating System Support

Added support for the following operating systems:

- Solaris 2.10
- Windows XP x64 (32-bit mode)

For more information, see the *Polyspace Installation Guide*.

Polyspace Model Link TL Product

Enhanced Installer

Version 5.1 includes an enhanced and simplified installer for all Polyspace products. The installation process is now faster and easier to complete than in previous releases.

For more information, see the *Polyspace Installation Guide*.

Operating System Support

Added support for the following operating systems:

- Solaris 2.10
- Windows XP x64 (32-bit mode)

For more information, see the *Polyspace Installation Guide*.

Polyspace UML Link RH Product

Enhanced Installer

Version 5.1 includes an enhanced and simplified installer for all Polyspace products. The installation process is now faster and easier to complete than in previous releases.

For more information, see the *Polyspace Installation Guide*.

Rhapsody Support

Added support for Telelogic Rhapsody Version 7.1.

C Language Support

Added support for C language in Rhapsody software.

For more information, see the *Polyspace UML Link RH User's Guide*.

Operating System Support

Added support for the following operating systems:

- Solaris 2.10
- Windows XP x64 (32-bit mode)

For more information, see the *Polyspace Installation Guide*.

Compatibility Summary for Polyspace Software

This table summarizes new features and changes that might cause incompatibilities when you upgrade from an earlier version, or when you use files on multiple versions. Details are provided in the description of the new feature or change.

Version (Release)	New Features and Changes with Version Compatibility Impact
Latest Version for C/C++ Products V8.1 (R2011a)	See the **Compatibility Considerations** subheading for these new features or changes: • "Overflow Check Customization" on page 12 • "Main Generator Improvements" on page 13 • "Precision Improvements" on page 41 • "Permissive Mode Set By Default" on page 15 • "Product Name Change in Files and Folders" on page 17 • "Changes to Verification Results" on page 18 • "Changes to Coding Rules Checker Results" on page 20
Latest Version for Ada Products V6.1 (R2011a)	See the **Compatibility Considerations** subheading for these new features or changes: • "Generated Main with Explicit Tasks and Accept Statements" on page 29 • "Enhancements in Run-Time Checks Perspective" on page 29 • "UOVFL and UNFL Checks Removed" on page 29 • "NIV Checks for Universal Constants" on page 30

Version (Release)	New Features and Changes with Version Compatibility Impact
	• "Scaling Issue for Large Applications with Nested Structures/Arrays" on page 31 • "Product Name Change in Files and Folders" on page 31 • "Changes to Verification Results" on page 32 • "Generated Main with Explicit Tasks and Accept Statements" on page 37
Latest Version for Model Link Products V5.7 (R2011a)	See the **Compatibility Considerations** subheading for this new feature or change: • "Overflow Check Customization" on page 39 • "Main Generator Improvements" on page 40 • "Precision Improvements" on page 41
V8.0 (R2010b) for C/C++ Products	See the **Compatibility Considerations** subheading for these new features or changes: • "New Options to Classify Run-Time Checks and Coding Rules Violations" on page 51 • "Main Generation in C++" on page 53 • "Default Target Processor" on page 56 • "Default Operating System Target" on page 56 • "Include Folders Added to Verification by Default" on page 56 • "Changes to Verification Results" on page 57

Version (Release)	New Features and Changes with Version Compatibility Impact
V6.0 (R2010b) for Ada Products	See the **Compatibility Considerations** subheading for these new features or changes: • "New Options to Classify Run-Time Checks" on page 72 • "Default Target Processor" on page 74
V5.6 (R2010b) for Model Link Products	See the **Compatibility Considerations** subheading for this new feature or change: • "Verification Options Set by Default" on page 78
V7.2 for C/C++ (R2010a)	See the **Compatibility Considerations** subheading for these new features or changes: • "Importing Review Comments" on page 82 • "Data Range Specifications (DRS) Enhancements" on page 83 • "Methodological Assistant Enhancements" on page 90 • "Class Analyzer Enhancements for C++" on page 91 • "Change to Time Format in Log File" on page 91 • "Merging of `OVFL` and `UNFL` Checks" on page 91 • "Improved `UNR` Checks" on page 92 • "Changes to Verification Results" on page 93 • "Changes to Coding Rules Checker Results" on page 101

Version (Release)	New Features and Changes with Version Compatibility Impact
V5.5 for Ada (R2010a)	See the **Compatibility Considerations** subheading for this new feature or change: • "Data Range Specifications for Custom Simulink Data Objects" on page 113
V7.1 for C/C++ (R2009b)	See the **Compatibility Considerations** subheading for this new feature or change: • "Changes to Coding Rules Checker Results" on page 117
V5.4 for Ada (R2009b)	See the **Compatibility Considerations** subheading for this new feature or change: • "Main Generator Enhancements" on page 121
V7.0 for C/C++ (R2009a)	See the **Compatibility Considerations** subheading for these new features or changes: • "Architecture Improvements" on page 130 • "Mathematical Functions Included in Stubs" on page 131 • "Automatic Orange Tester" on page 133
V5.3 for Ada (R2009a)	None
V6.0 for C/C++ (R2008b)	None
V5.2 for Ada (R2008b)	None
V5.1 (R2008a)	See the **Compatibility Considerations** subheading for this new feature or change: • "Removed Cygwin Software Dependency for Windows Platforms" on page 144