

Polyspace[®] Code Prover[™] Release Notes



MATLAB[®]&SIMULINK[®]

How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Polyspace[®] Code Prover[™] Release Notes

© COPYRIGHT 2013–2015 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

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

Trademarks

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

Patents

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

Simplified workflow for project setup and results review with a unified user interface	1-2
Review of code complexity metrics and global variable usage in user interface	1-3
Code Complexity Metrics	1-3
Global Variables	1-4
Context-sensitive help for code complexity metrics, MISRA-C:2012, and custom coding rules	1-5
Detection of stack pointer dereference outside scope	1-5
Review of latest results compared to the last run	1-6
Guidance for reviewing Polyspace Code Prover checks in C code	1-7
Improvements in search capability in the user interface ...	1-7
Isolated ellipsis for variable number of function arguments supported	1-8
Improvement in pointer comparisons	1-8
Improvements in coding rules checking	1-9
Simplified results infrastructure	1-11
Support for GCC 4.8	1-11

Polyspace plug-in for Simulink improvements	1-11
Integration with Simulink projects	1-11
DRS file format changed to XML	1-12
Back-to-model available when Simulink is closed	1-12
Polyspace binaries being removed	1-12
Import Visual Studio project being removed	1-13

R2014b

Support for MISRA C:2012	2-2
Improved verification speed	2-2
Support for Mac OS	2-3
Improved verification precision for non-initialized variables	2-3
Read Operations on Structures	2-3
Other Operations	2-5
Support for C++11	2-6
Context-sensitive help for verification options and checks .	2-6
Code Editor for editing source files in Polyspace user interface	2-7
Local file-by-file verification	2-7
Simulink plug-in support for custom project files	2-8
TargetLink support updated	2-8
AUTOSAR support added	2-8
New checks for functions not called	2-9

Default verification level changed	2-9
Improved precision level	2-10
Default mode changed for C++ code verification in user interface	2-11
Updated Software Quality Objectives	2-11
Improved global menu in user interface	2-11
Improved Project Manager perspective	2-12
Changed analysis options	2-13
Improved Results Manager perspective	2-13
Error mode removed from coding rules checking	2-15
Remote launcher and queue manager renamed	2-15
Polyspace binaries being removed	2-16
Import Visual Studio project being removed	2-17

R2014a

Automatic project setup from build systems	3-2
Support for GNU 4.7 and Microsoft Visual Studio C++ 2012 dialects	3-2
Documentation in Japanese	3-3
Support for additional Coding Rules (MISRA C:2004 Rule 18.2, MISRA C++ Rule 5-0-11)	3-3
Preferences file moved	3-3

Support for batch analysis security levels	3-3
Interactive mode for remote verification	3-4
Default text editor	3-4
Results folder appearance in Project Browser	3-4
Results Manager improvements	3-6
Simplification of coding rules checking	3-8
Support for Windows 8 and Windows Server 2012	3-9
Check model configuration automatically before analysis .	3-10
Additional back-to-model support for Simulink plug-in ...	3-10
Function replacement in Simulink plug-in	3-10
Polyspace binaries being removed	3-11
Improvement of floating point precision	3-11

R2013b

Proven absence of certain run-time errors in C and C++ code	4-2
Color-coding of run-time errors directly in code	4-2
Calculation of range information for variables, function parameters and return values	4-2
Identification of variables exceeding specified range limits	4-3
Quality metrics for tracking conformance to software quality objectives	4-3

Web-based dashboard providing code metrics and quality status	4-4
Guided review-checking process for classifying results and run-time error status	4-4
Graphical display of variable reads and writes	4-5
Comparison with R2013a Polyspace products	4-5

R2015a

Version: 9.3

New Features

Bug Fixes

Compatibility Considerations

Simplified workflow for project setup and results review with a unified user interface

In R2015a, the Project and Results Manager perspectives are now unified. You can run verification and review results without switching between two perspectives.

The major changes are:

- You can start a new verification during your results review. Previously, you started a new verification only from the Project Manager perspective.
- After a verification, the result opens automatically. If you are looking at a previous result when a verification is over, you can load the new result or retain the previous one on the **Results Summary** pane. If you retain the previous results, you can later open the new results from the **Project Browser**. The new results are highlighted.
- You can have any of the panes open in the unified interface.

Previously, you could open the following panes only in one of the two perspectives.

Project Manager	Results Manager
<ul style="list-style-type: none"> • Project Browser: Set up project. • Configuration: Specify analysis options for your project. • Output Summary: Monitor progress of verification. • Run Log: Find detailed information about a verification. 	<ul style="list-style-type: none"> • Results Summary: View Polyspace[®] results. • Source: View read-only form of source code color coded with Polyspace results. • Check Details: View details of a particular result. • Check Review: Comment on a particular result. • Variable Access: View global variables and read/write operations on them. • Call Hierarchy: View callers and callees of a function. • Results Properties: Same as Run Log, but associated with results instead of a project. This pane has been removed.

Project Manager	Results Manager
	<p>To open the log associated with a result, with the results open, select Window > Show/Hide View > Run Log.</p> <ul style="list-style-type: none"> • Settings: Same information as Configuration, but associated with results instead of a project. This pane has been removed. <p>To open the configuration associated with a result, with the results open, select Window > Show/Hide View > Configuration.</p> <ul style="list-style-type: none"> • Orange Sources: View sources of orange checks. • Sensitivity Context: For a check that has a different color for different function calls, view the check color for each function call.

Review of code complexity metrics and global variable usage in user interface

- “Code Complexity Metrics” on page 1-3
- “Global Variables” on page 1-4

Code Complexity Metrics

In R2015a, you can view code complexity metrics in the Polyspace user interface. For more information, see “Code Metrics”. Previously, this information was available only in the Polyspace Metrics web interface.

In the user interface, you can:

- Specify a limit for the value of a metric. If the metric value for your source code exceeds this limit, the metric appears red on the **Results Summary** pane.

- Justify the value of a metric. If a metric value exceeds specified limits and appears red, you can add a comment with the rationale.

Combining these actions, you can enforce coding standards across your organization. For more information, see “Review Code Metrics”.

Reducing the complexity of your code improves code readability, reduces the possibility of coding errors, and allows more precise Polyspace verification.

Global Variables

In R2015a, you can comment and justify global variable usage on the **Results Summary** pane. Previously, you viewed global variable usage on the **Variable Access** pane, but could not comment on them.

On the **Results Summary** pane, global variables are classified into one of the following categories.

Category		Color	Meaning
Shared	Potentially unprotected	Orange	Global variables shared between multiple tasks but possibly not protected from concurrent access by the tasks
	Protected	Green	Global variables shared between multiple tasks and protected from concurrent access by the tasks
Not shared	Used	Black	Global variables used in a single task
	Unused	Gray	Global variables declared but not used

For more information, see “Global Variables”.

For code that you do not intend for multitasking, all variables are nonshared and can be either used or unused. For code that you intend for multitasking, you can specify tasks and protections through the analysis options for multitasking. For more information, see “Multitasking”.

You can still view the global variables on the **Variable Access** pane.

- To comment and justify potentially unprotected and unused global variables, use the **Results Summary** pane.
- To find the read and write operations on a global variable, use the **Check Details** or **Variable Access** pane. On the **Variable Access** pane, you can also see the variable range and other information.

For more information, see “Review Global Variable Usage”.

Context-sensitive help for code complexity metrics, MISRA-C:2012, and custom coding rules

In R2015a, context-sensitive help is available in the user interface for code complexity metrics, MISRA C[®]:2012 rule violations, and custom coding rule violations.

To access the contextual help, see “Getting Help”.

For information about these results, see:

- “Code Metrics”
- “MISRA C:2012 Directives and Rules”
- “Custom Coding Rules”

Detection of stack pointer dereference outside scope


In R2015a, the **Illegally dereferenced pointer** check can detect stack pointer dereference outside scope. Such dereference can happen, for example, when a pointer to a variable that is local to a function is returned from the function. Because the scope of the variable is limited to the function, dereferencing the pointer outside the function can cause undefined behavior.

This enhancement is not available by default. Use the option `-detect-pointer-escape` to detect such dereferences. To provide command-line options in the user interface:

- 1 On the **Configuration** pane, select **Advanced Settings**.
- 2 Enter the option in the **Other** field.

Before R2015a	R2015a
<p>In the following code, <code>ptr</code> points to <code>ret</code>. Because the scope of <code>ret</code> is limited to <code>func1</code>, when <code>ptr</code> is accessed in <code>func2</code>, the access is illegal. Polyspace Code Prover™ did not detect such pointer escapes.</p> <pre>void func2(int *ptr) { *ptr = 0; } int* func1(void) { int ret = 0; return &ret ; } void main(void) { int* ptr = func1() ; func2(ptr) ; }</pre>	<p>In the following code, Polyspace Code Prover produces a red Illegally dereferenced pointer check on <code>*ptr</code>.</p> <pre>void func2(int *ptr) { *ptr = 0; } int* func1(void) { int ret = 0; return &ret ; } void main(void) { int* ptr = func1() ; func2(ptr) ; }</pre>

The **Check Details** pane displays a message indicating that `ret` is accessed outside its scope.

 **ID 1: Illegally dereferenced pointer**

Error: pointer is outside its bounds

This check may be a path-related issue, which is not dependent on input values

Dereference of parameter 'ptr' (pointer to int 32, size: 32 bits):

Pointer is not null.

Points to 4 bytes at offset 0 in buffer of 4 bytes, so is within bounds (if memory is allocated).

Pointer may point to variable or field of variable:

'ret', local to function 'func1'. 'ret' is accessed outside its scope.

Review of latest results compared to the last run

In R2015a, you can review only new results compared to the previous run.

If you rerun your verification, the new results are displayed with an asterisk (*) against them on the **Results Summary** pane. To filter only these new checks, select the **New results** box.


If you make changes in your source code, you can use this feature to see only the checks introduced due to those changes. You can avoid reviewing checks in the source code that you did not change.

Guidance for reviewing Polyspace Code Prover checks in C code

In R2015a, the context-sensitive help for checks provides guidance about how to review the check. The help describes:

- Information available in the software for the check.
- In your source code, how to navigate to the root cause of the check.
- Common causes of the check.

To open the context-sensitive help for a check:

- On the **Results Summary** or **Source** pane, select the check.
- Select the  button.
- Select the link in the section **Diagnosing This Check**.

This additional guidance is not available for C++-specific checks.

Improvements in search capability in the user interface

In R2015a, the **Search** pane allows you to search for a string in various panes of the user interface.

To search for a string in the new user interface:

- 1 If the **Search** pane is not visible, open it. Select **Window > Show/Hide View > Search**.
- 2 Enter your string in the search box.
- 3 From the drop-down list beside the box, select names of panes you want to search.

The **Search** pane consolidates the search options previously available.

Isolated ellipsis for variable number of function arguments supported

In R2015a, for C++ code, Polyspace Code Prover supports the ellipsis in the function definition syntax `void foo(...){}` to mean variable number of arguments. Previously, the use of ellipsis in isolation was not supported. You could use only the syntax where the ellipsis was preceded with other parameters.

Before R2015a	R2015a
<p>In the following code, Polyspace considers that <code>foo</code> has no arguments. Therefore, it produces a red Correctness condition error on the second function call. The Check Details pane indicates that the wrong number of arguments were used in the function call.</p> <pre data-bbox="246 777 546 999"> void foo(...) { /* Function body */ } void main() { foo(); foo(1,2); //Red COR } </pre>	<p>In the following code, Polyspace considers that <code>foo</code> takes a variable number of arguments. It does not produce a red Correctness condition error on the second function call.</p> <pre data-bbox="798 713 1095 939"> void foo(...) { /* Function body */ } void main() { foo(); foo(1,2); //No COR } </pre>

Improvement in pointer comparisons

In R2015a, Polyspace is more precise on pointer comparisons. In certain cases, if the software can determine that a pointer comparison is always true or false, it provides that result. Previously, Polyspace did not check pointer comparisons.

Before R2015a	R2015a
<p>In the following code, Polyspace does not check the comparison <code>ptr==&invalid</code>. Therefore, it considers that <code>check</code> can return either 0 or 1. In the <code>main</code> function, it verifies both branches of the <code>if-else</code> statement.</p> <pre data-bbox="246 1515 516 1538">#include <stdlib.h></pre>	<p>In the following code, Polyspace checks the comparison <code>ptr===&invalid</code> and determines that it is always true. Therefore, it considers that the <code>if</code> test is redundant and the function <code>check</code> returns 1 only. In the <code>main</code> function, it verifies the</p>

Before R2015a	R2015a
<pre> typedef unsigned char U8; U8 invalid; #define TEST_DISABLED &invalid U8 check(U8 cnt, U8* ptr) { U8 ret=0; if (ptr == &invalid) { ret=1; } return ret; } void main() { U8 isDisabled; isDisabled = check(1U,TEST_DISABLED); if(isDisabled == 1) { /* Do not perform test */ } else { /* Perform test */ } } </pre>	<pre> if branch and considers the else branch as unreachable. #include <stdlib.h> typedef unsigned char U8; U8 invalid; #define TEST_DISABLED &invalid U8 check(U8 cnt, U8* ptr) { U8 ret=0; if(ptr == &invalid) { ret=1; } return ret; } void main() { U8 isDisabled; isDisabled = check(1U,TEST_DISABLED); if(isDisabled == 1) { /* Do not perform test */ } else { /* Perform test */ } } </pre>

Improvements in coding rules checking

MISRA C:2004 and MISRA AC AGC

Rule Number	Effect	More Information
Rule 12.6	More results on noncompliant #if preprocessor directives Fewer results for variables cast to effective Boolean types.	MISRA C:2004 Rules — Chapter 12: Expressions
Rule 12.12	Fewer results when converting to an array of float	MISRA C:2004 Rules — Chapter 12: Expressions

MISRA C:2012

Rule Number	Effect	More Information
Rules 10.3	Fewer results on enumeration constants when the type of the constant is a named enumeration type. Fewer results on user-defined effective Boolean types.	MISRA C:2012 Rule 10.3
Rule 10.4	Fewer results on enumeration constants when the type of the constant is a named enumeration type. Fewer results for casts to user-defined effective Boolean types.	MISRA C:2012 Rule 10.4
Rule 10.5	Fewer results on enumeration constants when the type of the constant is a named enumeration type. Fewer results on user-defined effective Boolean types.	MISRA C:2012 Rule 10.5
Rule 12.1	More results on expressions with <code>sizeof</code> operator and on expressions with <code>?</code> operators. Fewer results on operators of the same precedence and in preprocessing directives.	MISRA C:2012 Rule 12.1
Rule 14.3	No results for non-controlling expressions.	MISRA C:2012 Rule 14.3

MISRA C++:2008

Rule Number	Effect	More Information
Rule 5-0-3	Fewer results on enumeration constants when the type of the constant is the enumeration type.	MISRA C++ Rules — Chapter 5
Rule 6-5-1	Fewer results on compliant vector variable iterators.	MISRA C++ Rules — Chapter 6

Rule Number	Effect	More Information
Rule 14-8-2	Fewer results for functions contained in the “Files and folders to ignore (C++)” option.	MISRA C++ Rules — Chapter 14
Rule 15-3-2	Fewer results for user-defined return statements after a <code>try</code> block.	MISRA C++ Rules — Chapter 15

Simplified results infrastructure

Polyspace results folders are reorganized and simplified. Files have been removed, combined, renamed, or moved. The changes do not affect the results that you see in the Polyspace environment.

Some important changes and file locations:

- The main results file is now encrypted and renamed `ps_results.pscp`. You can view results only in the Polyspace environment.
- The log file, `Polyspace_R2015a_project_date-time.log` has not changed.

For more information, see Results Folder Contents.

Support for GCC 4.8

Polyspace now supports the GCC 4.8 dialect for C and C++ projects.

To allow GCC 4.8 extensions in your Polyspace Code Prover verification, set **Target & Compiler > Dialect** option `gnu4.8`.

For more information, see “Dialect (C)” and “Dialect (C++)”.

Polyspace plug-in for Simulink improvements

In R2015a, there are three improvements to the Polyspace Simulink[®] plug-in.

Integration with Simulink projects

You can now save your Polyspace results to a Simulink project. Using this feature, you can organize and control your Polyspace results alongside your model files and folders.

To save your results to a Simulink project:

- 1 Open your Simulink project.
- 2 From your model, select **Code > Polyspace > Options**.
- 3 In the Polyspace parameter configuration tab, select the **Save results to Simulink project** option.

For more information, see “Save Results to a Simulink Project”.

DRS file format changed to XML

By default, the DRS files generated in Simulink are saved in XML.

For more information, see “XML File Format for Constraints”

If you want to use a customized .txt DRS file, contact customer support.

Back-to-model available when Simulink is closed

In the Polyspace plug-in for Simulink, the back-to-model feature now works even when your model is closed. When you click a link in your Polyspace results, MATLAB® opens your Simulink model and highlights the appropriate block.

Note: This feature works only with Simulink R2013b and later.

For more information about the back-to-model feature, see “Identify Errors in Simulink Models”.

Polyspace binaries being removed

The following Polyspace binaries will be removed in a future release. The binaries are located in *matlabroot/polyspace/bin*. You get a warning if you run them.

Binary name	Use instead
polyspace-automatic -orange-tester.exe	From the Polyspace environment, select Tools > Automatic Orange Tester
polyspace-c.exe	polyspace-code-prover-nodesktop -lang c

Binary name	Use instead
polyspace-cpp.exe	polyspace-code-prover-nodesktop -lang cpp
polyspace-remote-c.exe	polyspace-code-prover-nodesktop -lang c -batch
polyspace-remote-cpp.exe	polyspace-code-prover-nodesktop -lang cpp -batch
polyspace-remote.exe	polyspace-code-prover-nodesktop -batch
polyspace-rl-manager.exe	polyspace-server-settings.exe
polyspace-spooler.exe	polyspace-job-monitor.exe
polyspace-ver.exe	polyspace-code-prover-nodesktop -ver

Import Visual Studio project being removed

The **File > Import Visual Studio project** will be removed in a future release. Instead, use the **Create from build system** option during New Project creation. For more information, see “Trace Visual Studio Build”.

R2014b

Version: 9.2

New Features

Bug Fixes

Compatibility Considerations

Support for MISRA C:2012

Polyspace can now check your code against MISRA C:2012 directives and coding rules. To check for MISRA C:2012 coding rule violations:

- 1 On the **Configuration** pane, select **Coding Rules**.
- 2 Select **Check MISRA C:2012**.
- 3 The MISRA C:2012 guidelines have different categories for handwritten and automatically generated code.

If you want to use the settings for automatically generated code, also select **Use generated code requirements**.

For more information about supported rules, see MISRA C:2012 Coding Directives and Rules.

Improved verification speed

In R2014b, the following two changes improve the verification speed:

- Polyspace Code Prover can run the compilation phase of your verification in parallel on multiple processors. The software detects available processors and uses them to compile different source files in parallel.

Previously, the software ran post-compilation phases in parallel but compiled the source files sequentially. Starting in R2014b, the software can use multiple processors for the entire verification process.

To explicitly specify the number of processors, use the command-line option `-max-processes`. For more information, see `-max-processes`.

- Polyspace Code Prover has an improved engine for verification. This engine typically improves verification speed by 25%. However, in some cases, verification can take the same amount of time or longer.

Compatibility Considerations

In most cases, you do not see significant change in the number of checks resulting from the improved engine. If you see a major increase in the number of orange checks, contact technical support. For more information, see Obtain System Information for Technical Support.

Support for Mac OS

You can install and run Polyspace on Mac OS X. Polyspace is supported for Mac OS 10.7.4+, 10.8, and 10.9.

You can use Polyspace Metrics on Safari and set up your Mac as a Metrics server. However, if you restart your Mac machine that is setup as a Metrics server, you must restart the Polyspace server daemon.

The Automatic Orange Tester is not supported for Mac.

Improved verification precision for non-initialized variables

Polyspace Code Prover performs the following checks for initialization:

- Non-initialized local variable or NIVL
- Non-initialized variable or NIV

In R2014b, the following changes appear in these checks.

Read Operations on Structures

When you read structured variables, Polyspace Code Prover performs a check for initialization. This check helps detect partially initialized and non-initialized structures earlier in the code.

Prior to R2014b	R2014b
<ul style="list-style-type: none">• When you read structured variable, a check for initialization was not performed.• The checks occurred only when you read individual fields of a structured variable, provided the fields themselves were not structured variables.	<p>When you read structured variables, a check for initialization occurs. The check turns:</p> <ul style="list-style-type: none">• Green, if all fields of the structure that are used are initialized. If no field is used, the check is green by default.• Red, if all fields that are used are not initialized.• Orange, if only some fields that are used are initialized. Following the check, Polyspace considers that the

Prior to R2014b	R2014b
	<p>uninitialized fields have the full range of values allowed by their type.</p> <p>Polyspace considers a field as used if there is a read or write operation on the field anywhere in the code. Polyspace does not check for initialization of fields that are not used.</p> <p>To determine which fields Polyspace checked for initialization:</p> <ol style="list-style-type: none"><li data-bbox="798 661 1299 756">1 Select the NIV or NIVL check on the Results Summary pane or Source pane.<li data-bbox="798 770 1248 829">2 View the message on the Check Details pane.

Prior to R2014b	R2014b
<p>Example:</p> <pre>typedef struct S { int a; int b; }S; void func1(S); void func2(int); void main() { S varS; func1(varS); func2(varS.a); }</pre> <p>A check was not performed when the non-initialized structure <code>varS</code> was read. When the field <code>a</code> of <code>varS</code> was read, a red NIVL check appeared.</p>	<p>Example:</p> <pre>typedef struct S { int a; int b; }S; void func1(S); void func2(int); void main() { S varS; func1(varS); func2(varS.a); }</pre> <p>When the non-initialized structure <code>varS</code> is read, a red NIVL check appears.</p> <p>For more examples, see:</p> <ul style="list-style-type: none"> • Partially initialized structure — All used fields initialized • Partially initialized structure — Some used fields initialized

Other Operations

The specification of **Non-initialized variable** checks has changed for the following operations. These operations are not commonly used. Therefore, it is likely that these changes do not affect your Polyspace verification.

Prior to R2014b	R2014b
<p>If you initialized only the high bits of a variable through a pointer, an orange check for initialization appeared when the variable was read.</p>	<p>If you initialize only the high bits of a variable through a pointer, a green check for initialization appears when the variable is read.</p>
<p>If you performed an operation on a C++ object after it was destroyed, a red check for initialization appeared on the operation.</p>	<p>If you perform an operation on a C++ object after it is destroyed, the check for initialization has the same color as</p>

Prior to R2014b	R2014b
The check indicated that the object was destroyed.	before the destruction. Polyspace does not introduce a red check on this type of access.

Compatibility Considerations

If you use an earlier version of Polyspace Code Prover, it is possible that you see the following changes in your results.

- Read operation on structures: You see an increase in the total number of checks.

However, some red or orange NIV or NIVL checks on the fields of structures turn green. Instead, you see some new red or orange checks on the structures themselves.
- Other operations:
 - If you have operations that initialize only the high bits of a variable through a pointer, you can see a reduction in orange NIV or NIVL checks.
 - If you have operations that access an object after it is destroyed, you can see a reduction in red NIV or NIVL checks.

Support for C++11

Polyspace can now fully analyze C++ code that follows the ISO/IEC 14882:2011 standard, also called C++11.

Use two new analysis options when analyzing C++11 code. On the **Target & Compiler** pane, select:

- **C++11 extensions** to allow the standard C++11 libraries and functions during your analysis.
- **Block char 16/32_t types** to not allow char16_t or char32_t types during the analysis.

For more information, see C++11 Extensions (C++) and Block char16/32_t types (C++).


Context-sensitive help for verification options and checks

In R2014b, contextual help is available for verification options in the Polyspace interface and its plug-ins. To view the contextual help:

-
- 1 Hover your cursor over a verification option in the **Configuration** pane.
 - 2 Inside the tooltip, select the “More Help” link.

The documentation for that option appears in a dockable window.

Contextual help is available in the Polyspace interface for run-time errors. To view the contextual help for checks:

- 1 In the Results Manager perspective, select a run-time error from the results.
- 2 Inside the **Check Details** pane, select .

The documentation for that check appears in a docked window.

For more information, see Getting Help.

Code Editor for editing source files in Polyspace user interface

In R2014b, by default, you can edit your source files inside the Polyspace user interface.

- In the Project Manager perspective, on the **Project Browser** tree, double-click your source file.
- In the Results Manager perspective, right-click the **Source** pane and select **Open Source File**.

Your source files appear on a **Code Editor** tab. On this tab, you can edit your source files and save them.

To use an external text editor, change your preferences.

- 1 Select **Tools > Preferences**.
- 2 Specify an external editor on the **Editors** tab.

For more information, see Specify External Text Editor.

Local file-by-file verification

In R2014b, you can verify your source code file by file on your local installation of Polyspace Code Prover. Each file is verified independently of the other files in your module. Previously, you performed file-by-file verification only on a remote server. The verification required:

- Parallel Computing Toolbox™ on the client side
- MATLAB Distributed Computing Server™ on the server side

For more information on file-by-file verification, see:

- Run File-by-File Verification
- Open Results of File-by-File Verification

For information on file-by-file verification in batch mode, see:

- Run File-by-File Batch Verification
- Open Results of File-by-File Batch Verification

Simulink plug-in support for custom project files

With the Polyspace plug-in for Simulink, you can now use a project file to specify the verification options.

On the **Polyspace** pane of the Configuration Parameters window, with the **Use custom project file** option you can enter a path or browse for a `.psprj` project file.

For more information, see [Configure Polyspace Analysis Options](#).

TargetLink support updated

The Polyspace plug-in for Simulink now supports TargetLink® 3.4 and 3.5. Older versions of TargetLink are not supported.

For more information, see [TargetLink Considerations](#).

AUTOSAR support added

In R2013b, the Polyspace plug-in for Simulink added support for AUTOSAR generated code with Embedded Coder®. If you use `autosar.tlc` as your **System target file** for code generation, when you run Polyspace, the verification can use the data range information from AUTOSAR.

The Polyspace verification uses the same default options and parameters as it does for Embedded Coder.

For more information, see *Embedded Coder Considerations*.

New checks for functions not called

Two new checks in Polyspace Code Prover detect C/C++ functions that are defined but not called during execution of the code.

Check	Purpose
Function not called	Detects functions that are defined but not called in the source files.
Function not reachable	Detects functions that are defined but called only from an unreachable part of the source.

You can choose to activate these checks using the following options:

- In the user interface, on the **Configuration** pane, under **Check Behavior**, select a value for the option **Detect uncalled functions**.
- At the command line, use the option `-uncalled-function-checks` with an appropriate argument.

Goal	Option Value
Do not detect uncalled functions.	none
Detect functions that are defined but not called.	never-called
Detect functions that are defined and called only from an unreachable part of the code.	called-from-unreachable
Detect all uncalled functions.	all

Default verification level changed

In R2014b, unless you specify a verification level explicitly, Polyspace Code Prover verification performs two passes on your source code instead of four. For instance:

- In the user interface, on the **Output Summary** tab, you can see that the verification continues to **Level2**. For more passes, on the **Configuration** pane, under the **Precision** node, select a higher **Verification level**.

- At the command line, the verification implicitly uses `-to pass2`. For more passes, use the `-to` option explicitly with a higher pass value.

The default verification is completed in much less time.

For more information, see:

- Verification level (C)
- Verification level (C++)

Compatibility Considerations

If you do not specify a verification level explicitly in your `polyspace-code-prover-nodesktop` command, your verification runs to **Software Safety Analysis Level 2**. In most cases, this verification level produces only slightly more orange checks than **Software Safety Analysis Level 4**. However, if you see a significant change in your results, to reproduce your earlier results:

- In the user interface, select **Software Safety Analysis Level 4** for **Verification level**.
- At the command line, use the option `-to pass4` with the `polyspace-code-prover-nodesktop` command.

Improved precision level

In R2014b, certain internal limits have been removed from verification that uses a **Precision level** of 3. Because of this improvement, you can use this **Precision level** to significantly reduce orange checks, especially for multitasking code that uses shared variables. However, if you use this level, the verification can take significantly longer.

To set **Precision level** to 3, do one of the following:

- In the user interface, on the **Configuration** pane, select **Precision**. From the **Precision level** drop-down list, select 3.
- At the DOS or UNIX[®] command prompt, use the flag `-O3` with the `polyspace-code-prover-nodesktop` command.
- At the MATLAB command prompt, use the argument `'-O3'` with the `polyspaceCodeProver` function.

For more information, see Precision level (C/C++).

Default mode changed for C++ code verification in user interface

When you create a new Polyspace Code Prover project with C++ as the project language, the following options are selected in the user interface by default. The options appear on the **Configuration** pane under the **Code Prover Verification** node.

Option	Value
Verify Module	On
Class	all
Functions to call within the specified classes	unused
Functions to call	unused
Variables to initialize	uninit

These options replace the default selection of **Verify whole application** on the Polyspace user interface.

If your C++ code does not contain a `main` function, Polyspace generates a `main` by default during verification from the user interface.

For more information on the `main` generation options, see [Provide Context for C++ Code Verification](#).

Updated Software Quality Objectives

In R2014b, the Software Quality Objectives or SQOs have been updated to include MISRA[®] C++: 2008 coding rule violations.

Using the predefined SQO levels, you can specify quality thresholds for your project or individual files in your project. With the updated SQOs, you can now specify that your project must not violate certain MISRA C++ rules.

For more information, see [Predefined SQO Levels](#).

Improved global menu in user interface

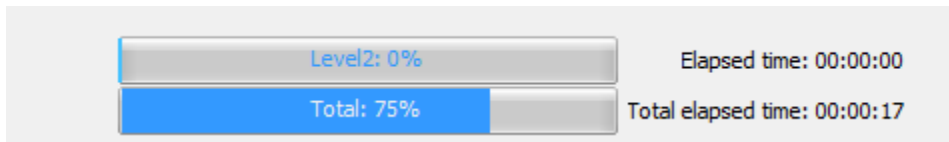
The global menu in the Polyspace user interface has been updated. The following table lists the current location for the existing global menu options.


Goal	Prior to R2014b	R2014b
Open the Polyspace Metrics interface in your web browser.	File > Open Metrics Web Interface	Metrics > Open Metrics
Upload results from the Polyspace user interface to Polyspace Metrics.	File > Upload in Polyspace Metrics repository	Metrics > Upload to Metrics
Update results stored in Polyspace Metrics with your review comments and justifications.	File > Save in Polyspace Metrics repository	Metrics > Save comments to Metrics
Generate a report from results after verification.	Run > Run Report > Run Report	Reporting > Run Report
Open generated report.	Run > Run Report > Open Report	Reporting > Open Report
Partition source code into modules.	Run > Run Modularize	Tools > Run Modularize
Import review comments from previous verification.	Review > Import	Tools > Import Comments
Specify code generator for generated code.	Review > Code Generator Support	Tools > Code Generator Support
Specify settings that apply to all Polyspace Code Prover projects.	Options > Preferences	Tools > Preferences
Specify settings for remote verification.	Options > Metrics and Remote Server Settings	Metrics > Metrics and Remote Server Settings

Improved Project Manager perspective

The following changes have been made in the Project Manager perspective:

- The **Progress Monitor** tab does not exist anymore. Instead, after you start a verification, you can view its progress on the **Output Summary** tab.
- Instead of a single progress bar showing all the stages of verification, you can see two progress bars. The top bar shows progress in the current stage of verification and the lower bar shows overall progress.



After verification, you can see the overall time taken. To see the time taken in each stage of verification, click the  icon.

- In the **Project Browser**, projects appear sorted in alphabetical order instead of order of creation.

Changed analysis options

Changes have been made to the following analysis options:

- On the **Configuration** pane, the analysis option **Files and folders to ignore** has been moved from **Coding Rules Checking** to **Inputs & Stubbing**. The functionality in Polyspace Code Prover has not changed.
- On the **Configuration** pane, the **Interactive** option has been removed from the graphical interface. To use interactive mode, use the `-interactive` flag at the command line or in the **Advanced Settings > Other** text field.
- You cannot use batch mode or interactive mode with **Verification Level > C/C++ source compliance checking**.

To run only to code compliance, run Polyspace Code Prover locally.

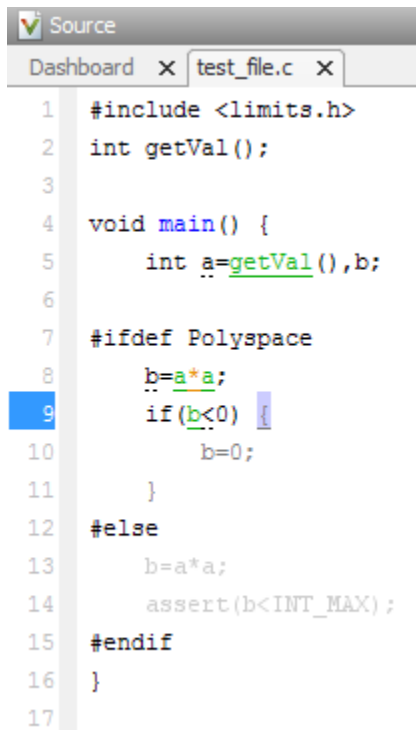
To perform batch or interactive verifications, use **Software Safety Analysis level 0** or higher.

Improved Results Manager perspective

The following changes have been made in the Results Manager perspective:

- On the **Source** pane, the following code appears in gray:
 - Code deactivated due to conditional compilation. Polyspace assigns a lighter shade of gray to this code.
 - Code in an unreachable branch. Polyspace assigns a darker shade of gray to this code.

For the difference between the two cases, see the code below. To reproduce the colors, before verification, on the **Configuration** pane, enter **Polyspace=** for **Preprocessor definitions**.



```
Source
Dashboard x test_file.c x
1 #include <limits.h>
2 int getVal();
3
4 void main() {
5     int a=getVal(),b;
6
7     #ifdef Polyspace
8         b=a*a;
9         if(b<0) {
10             b=0;
11         }
12     #else
13         b=a*a;
14         assert(b<INT_MAX);
15     #endif
16 }
17
```

- To prioritize your orange check review, use the **Show** menu on the **Results Summary** pane. This menu replaces the previously available methodologies for the same purpose.
 - To display red, gray, and orange checks likely to be run-time errors, from the **Show** menu, select **Critical checks**. This option replaces the **First checks to review** methodology.
 - To display all checks, from the **Show** menu, select **All checks**. This option replaces the **All checks** methodology.
 - The methodologies **Methodology for C/C++ > Light** and **Methodology for C/C++ > Moderate** have been removed.

-
- To create your own subset of orange checks to review, select **Tools > Preferences**. On the **Review Scope** tab, specify the number or percentage of orange checks of each type to review. The options on this tab replace the options on the **Review Configuration** tab.
 - To group your checks, use the **Group by** menu on the **Results Summary** pane.
 - To leave your checks ungrouped, instead of **List of Checks**, select **Group by > None**.
 - To group checks by check color and type, instead of **Checks by Family**, select **Group by > Family**.
 - To group checks by file and function, instead of **Checks by File/Function**, select **Group by > File**.
 - To view the percentage of checks that you have justified, instead of the **Review Statistics** pane, use the **Justified** column on the **Results Summary** pane. On this pane:
 - To view the percentage of checks that you justified broken down by color/type, select **Group by > Family**.
 - To view the percentage of checks that you justified broken down by file/function, select **Group by > File**.

Error mode removed from coding rules checking

In R2014b, the **Error** mode has been removed from coding rules checking. Therefore, coding rule violations cannot stop a verification.

Compatibility Considerations

For existing coding rules files, rules having the keyword **error** are treated in the same way as the keyword **warning**. For more information on **warning**, see **Format of Custom Coding Rules File**.

Remote launcher and queue manager renamed

Polyspace has renamed the remote launcher and the queue manager.

Previous name	New Name	More information
polyspace-rl-manager.exe	polyspace-server-settings.exe	Only the binary name has changed. The interface title, Metrics and Remote Server Settings , is unchanged.
polyspace-spooler.exe Queue Manager or Spooler	polyspace-job-monitor.exe Job Monitor	The binary and the interface titles have changed. Interface labels have changed in the Polyspace interface and its plug-ins.
pslinkfun('queuemanager')	pslinkfun('jobmonitor')	See pslinkfun.

Compatibility Considerations

If you use the old binaries or functions, you receive a warning.

Polyspace binaries being removed

The following Polyspace binaries will be removed in a future release. Unless otherwise noted, the binaries to use are located in *MATLAB Install/polyspace/bin*.

Binary name	What happens	Use instead
polyspace-automatic -orange-tester.exe	Warning	From the Polyspace environment, select Tools > Automatic Orange Tester
polyspace-c.exe	Warning	polyspace-code-prover-nodesktop -lang c
polyspace-cpp.exe	Warning	polyspace-code-prover-nodesktop -lang cpp
polyspace-remote-c.exe	Warning	polyspace-code-prover-nodesktop -lang c -batch
polyspace-remote-cpp.exe	Warning	polyspace-code-prover-nodesktop -lang cpp -batch

Binary name	What happens	Use instead
polyspace-remote.exe	Warning	polyspace-code-prover-nodesktop - batch
polyspace-rl-manager.exe	Warning	polyspace-server-settings.exe
polyspace-spooler.exe	Warning	polyspace-job-monitor.exe
polyspace-ver.exe	Warning	polyspace-code-prover-nodesktop -ver
setup-remote-launcher.exe	Warning	<i>MATLAB install</i> /toolbox/polyspace / psdistcomp/bin/setup-polyspace-cluster

Import Visual Studio project being removed

The **File > Import Visual Studio project** will be removed in a future release. Instead, use the **Create from build system** option during New Project creation. For more information, see Trace Visual Studio Build.

R2014a

Version: 9.1

New Features

Bug Fixes

Compatibility Considerations


Automatic project setup from build systems

In R2014a, you can set up a Polyspace project from build automation scripts that you use to build your software application. The automatic project setup runs your automation scripts to determine:

- Source files.
- Includes.
- **Target & Compiler** options.

To set up a project from your build automation scripts:

- On the DOS or UNIX command line: Use the `polyspace-configure` command. For more information, see [Create Project from DOS and UNIX Command Line](#).
- In the user interface: When creating a new project, in the **Project – Properties** window, select **Create from build command**. In the following window, enter:
 - The build command that you use.
 - The directory from which you run your build command.
 - Additional options. For more information, see [Create Project in User Interface](#).

Click . In the **Project Browser**, you see your new Polyspace project with the required source files, include folders, and **Target & Compiler** options.

- On the MATLAB command line: Use the `polyspaceConfigure` function. For more information, see [Create Project from MATLAB Command Line](#).

Support for GNU 4.7 and Microsoft Visual Studio C++ 2012 dialects

Polyspace supports two additional dialects: Microsoft® Visual Studio® C++ 2012 and GNU® 4.7. If your code uses language extensions from these dialects, specify the corresponding analysis option in your configuration. From the **Target & Compiler > Dialect** menu, select:

- `gnu4.7` for GNU 4.7
- `visual11.0` for Microsoft Visual Studio C++ 2012

For more information about these and other supported dialects, see [Dialects for C](#) or [Dialects for C++](#).

Documentation in Japanese

The Polyspace product, including the documentation, is available in Japanese.

To view the Japanese version of Polyspace Code Prover documentation, go to <http://www.mathworks.co.jp/jp/help/codeprover/>. If the documentation appears in English, from the country list beside the globe icon at the top of the page, select Japan.

Support for additional Coding Rules (MISRA C:2004 Rule 18.2, MISRA C++ Rule 5-0-11)

The Polyspace coding rules checker now supports two additional coding rules: MISRA C 18.2 and MISRA C++ 5-0-11.

- MISRA C 18.2 is a required rule that checks for assignments to overlapping objects.
- MISRA C++ 5-0-11 is a required rule that checks for the use of the plain `char` type as anything other than storage or character values.
- MISRA C++ 5-0-12 is a required rule that checks for the use of the signed and unsigned `char` types as anything other than numerical values.

For more information, see MISRA C:2004 Coding Rules or MISRA C++ Coding Rules.

Preferences file moved

In R2014a, the location of the Polyspace preferences file has been changed.

Operating System	Location before R2014a	Location in R2014a
Windows®	%APPDATA%\Polyspace	%APPDATA%\MathWorks\MATLAB\R2014a\Polyspace
Linux®	/home/\$USER/.polyspace	/home/\$USER/.matlab/\$RELEASE/Polyspace

For more information, see Storage of Polyspace Preferences.

Support for batch analysis security levels

When creating an MDCS server for Polyspace batch analyses, you can now add additional security levels through the **MATLAB Admin Center**. Using the **Metrics and Remote Server Settings**, the MDCS server is automatically set to security level

zero. If you want additional security for your server, use the **Admin Center** button. The additional security levels require authentication by user name, cluster user name and password, or network user name and password.

For more information, see MDCS documentation.

Interactive mode for remote verification

In R2014a, you can select an additional **Interactive** mode for remote verification. In this mode, when you run Polyspace Code Prover on a cluster, your local computer is tethered to the cluster through Parallel Computing Toolbox and MATLAB Distributed Computing Server.

To run verification in this mode

- In the user interface: On the **Configuration** pane, under **Distributed Computing**, select **Interactive**.
- On the DOS or UNIX command line, append `-interactive` to the `polyspace-code-prover-nodesktop` command.
- On the MATLAB command line, add the argument `'-interactive'` to the `polyspaceCodeProver` function.

For more information, see [Interactive](#).

Default text editor

In R2014a, Polyspace uses a default text editor for opening source files. The editor is:

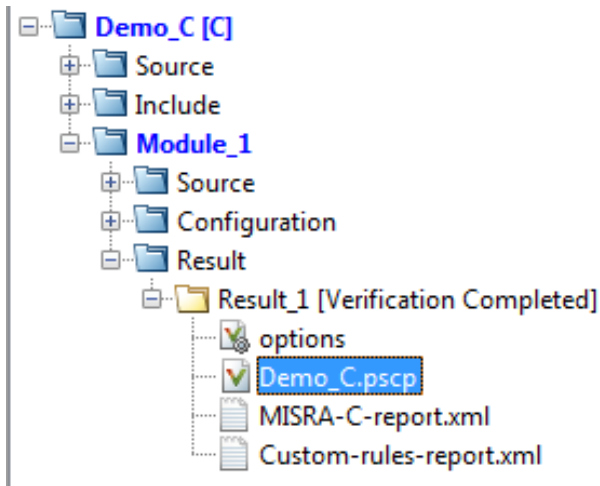
- WordPad in Windows
- vi in Linux

You can change the text editor on the **Editors** tab under **Options > Preferences**. For more information, see [Specify Text Editor](#).

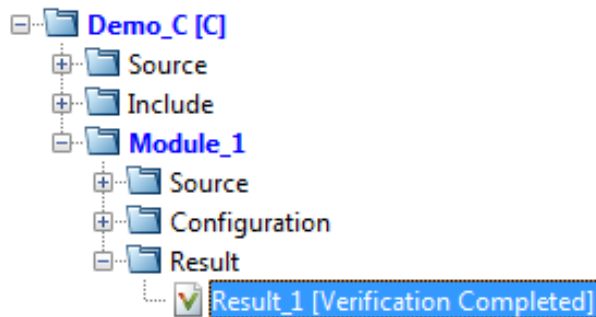
Results folder appearance in Project Browser

In R2014a, the results folder appears in a simplified form in the **Project Browser**. Instead of a folder containing several files, the result appears as a single file.

- Format before R2014a:



- Format in R2014a:



The following table lists the changes in the actions that you can perform on the results folder.

Action	2013b	2014a
Open results.	In the result folder, double-click result file with extension .pscp.	Double-click result file.
Open analysis options used for result.	In the result folder, select options .	Right-click result file and select Open Configuration .

Action	2013b	2014a
Open metrics page for batch analyses if you had used the analysis option Distributed Computing > Add to results repository .	In the result folder, select Metrics Web Page .	Double-click result file. If you had used the option Distributed Computing > Add to results repository , double-clicking the results file for the first time opens the metrics web page instead of the Result Manager perspective.
Open results folder in your file browser.	Navigate to results folder. To find results folder location, select Options > Preferences . View result folder location on the Project and Results Folder tab.	Right-click result file and select Open Folder with File Manager .

Results Manager improvements

- In R2014a, you can view the extent of a code block on the **Source** pane by clicking either its opening or closing brace.

```
Source
Dashboard x tasks1.c x
47 static void initregulate(void)
48 {
49     int tmp = 0;
50     while (random_int() < 1000) {
51         tmp = orderregulate();
52         Begin_CS();
53         tmp = SHR + SHR2 + SHR6;
54         End_CS();
55         tmp = Get_PowerLevel();
56         Compute_Injection();
57     } /* end loop; */
58 }
```

Source | Data Range Configuration

Note: This action does not highlight the code block if the brace itself is already highlighted. The opening brace can be highlighted, for instance, if there is an **Unreachable code** error on the code block.

- In R2014a, the **Verification Statistics** pane in the Project Manager and the **Results Statistics** pane in the Results Manager have been renamed **Dashboard**.

On the **Dashboard**, you can obtain an overview of the results in a graphical format. For more information, see [Dashboard](#).

- In R2014a, on the **Results Summary** pane, you can distinguish between violations of predefined coding rules such as MISRA C or C++ and custom coding rules.

- The predefined rules are indicated by ▼ .
- The custom rules are indicated by ▼ .

In addition, when you click on the **Check** column header on the **Results Summary** pane, the rules are sorted by rule number instead of alphabetically.

- In R2014a, you can double-click a variable name on the **Source** pane to highlight all instances of the variable.

Simplification of coding rules checking

In R2014a, the **Error** mode has been removed from coding rules checking. This mode applied only to:

- The option **Custom** for:
 - **Check MISRA C rules**
 - **Check MISRA AC AGC rules**
 - **Check MISRA C++ rules**
 - **Check JSF C++ rules**
- **Check custom rules**

The following table lists the changes that appear in coding rules checking.

Coding Rules Feature	2013b	2014a
New file wizard for custom coding rules.	<p>For each coding rule, you can select three results:</p> <ul style="list-style-type: none"> • Error: Analysis stops if the rule is violated. <p>The rule violation is displayed on the Output Summary tab in the Project Manager perspective.</p> <ul style="list-style-type: none"> • Warning: Analysis continues even if the rule is violated. 	<p>For each coding rule, you can select two results:</p> <ul style="list-style-type: none"> • On: Analysis continues even if the rule is violated. <p>The rule violation is displayed on the Results Summary pane in the Result Manager perspective.</p> <ul style="list-style-type: none"> • Off: Polyspace does not check for violation of the rule.

Coding Rules Feature	2013b	2014a
	<p>The rule violation is displayed on the Results Summary pane in the Result Manager perspective.</p> <ul style="list-style-type: none"> • Off: Polyspace does not check for violation of the rule. 	
Format of the custom coding rules file.	<p>Each line in the file must have the syntax:</p> <pre><i>rule</i> off error warning #<i>comments</i></pre> <p>For example:</p> <pre># MISRA configuration - Proj1 10.5 off #don't check 10.5 17.2 error 17.3 warning</pre>	<p>Each line in the file must have the syntax:</p> <pre><i>rule</i> off warning #<i>comments</i></pre> <p>For example:</p> <pre># MISRA configuration - Proj1 10.5 off #don't check 10.5 17.2 warning 17.3 warning</pre>

Compatibility Considerations

For existing coding rules files that use the keyword **error**:

- If you run analysis from the user interface, it will be treated in the same way as the keyword **warning**. The verification will not stop even if the rule is violated. The rule violation will however be reported on the **Results Summary** pane.
- If you run analysis from the command line, the verification will stop if the rule is violated.

Support for Windows 8 and Windows Server 2012

Polyspace supports installation and analysis on Windows Server[®] 2012 and Windows 8.

For installation instructions, see Installation, Licensing, and Activation.

Check model configuration automatically before analysis

For the Polyspace Simulink plug-in, the **Check configuration** feature has been enhanced to automatically check your model configuration before analysis. In the **Polyspace** pane of the Model Configuration options, select:

- **On, proceed with warnings** to automatically check the configuration before analysis and continue with analysis when only warnings are found.
- **On, stop for warnings** to automatically check the configuration before analysis and stop if warnings are found.
- **Off** to never check the configuration automatically before an analysis.

If the configuration check finds errors, Polyspace always stops the analysis.

For more information about **Check configuration**, see [Check Simulink Model Settings](#).

Additional back-to-model support for Simulink plug-in

As you click the different links, the corresponding block is highlighted in the model. Because of internal improvements, the back-to-model feature is more stable. Additionally, support has been added for Stateflow[®] charts in Target Link and Linux operating systems.

For more information about the back-to-model feature, see [Identify Errors in Simulink Models](#).

Function replacement in Simulink plug-in

The following functions have been replaced in the Simulink plug-in by the function `pslinkfun`. These functions be removed in a future release.

Function	What Happens?	Use This Function Instead
<code>PolyspaceAnnotation</code>	Warning	<code>pslinkfun('annotations',...)</code>
<code>PolySpaceGetTemplateCFGFile</code>	Warning	<code>pslinkfun('gettemplate')</code>
<code>PolySpaceHelp</code>	Warning	<code>pslinkfun('help')</code>
<code>PolySpaceEnableCOMServer</code>	Warning	<code>pslinkfun('enablebacktomodel')</code>

Function	What Happens?	Use This Function Instead
PolySpaceSpooler	Warning	pslinkfun('queuemanager')
PolySpaceViewer	Warning	pslinkfun('openresults',...)
PolySpaceSetTemplateCFGFile	Warning	pslinkfun('settemplate',...)
PolySpaceConfigure	Warning	pslinkfun('advancedoptions')
PolySpaceKillAnalysis	Warning	pslinkfun('stop')
PolySpaceMetrics	Warning	pslinkfun('metrics')

Polyspace binaries being removed

The following Polyspace binaries will be removed in a future release:

- polyspace-automatic-orange-tester.exe
- polyspace-c.exe
- polyspace-cpp.exe
- polyspace-modularize.exe
- polyspace-remote-c.exe
- polyspace-remote-cpp.exe
- polyspace-remote.exe
- polyspace-report-generator.exe
- polyspace-results-repository.exe
- polyspace-rl-manager.exe
- polyspace-spooler.exe
- polyspace-ver.exe
- setup-remote-launcher.exe

Improvement of floating point precision

In R2013b, Polyspace improved the precision of floating point representation. Previously, Polyspace represented the floating point values with intervals, as seen in the tooltips. Now, Polyspace uses a rounding method.

For example, the verification represents `float arr = 0.1`; as,

- Pre-R2013b, `arr = [9.9999E^-2, 1.0001E-1]`.
- Now, `arr = 0.1`.

R2013b

Version: 9.0

New Features

Proven absence of certain run-time errors in C and C++ code

Use Polyspace Code Prover to prove the absence of overflow, divide-by-zero, out-of-bounds array access, and certain other run-time errors in source code. To verify code, the software uses formal methods-based abstract interpretation techniques. The code verification is static. It does not require program execution, code instrumentation, or test cases. Before compilation and test, you can verify handwritten code, generated code, or a combination of these two types of code.

Color-coding of run-time errors directly in code

Polyspace Code Prover uses color coding to indicate the status of code elements.

- **Green** — Proved to never have a run-time error.
- **Red** — Proved to always have a run-time error.
- **Gray** — Proved to be unreachable, which can indicate a functional issue.
- **Orange** — Unproven, and can have an error.

Errors detected include:

- Overflows, underflows, divide-by-zero, and other arithmetic errors
- Out-of-bounds array access and illegally dereferenced pointers
- Always true/false statement due to dataflow propagation
- Read access operation on uninitialized data
- Dead code
- Access to null `this` pointer (C++)
- Dynamic errors related to object programming, inheritance, and exception handling (C++)
- Uninitialized class members (C++)
- Unsound type conversions

For more information, see Interpret Results.

Calculation of range information for variables, function parameters and return values

Polyspace Code Prover calculates and displays range information associated with, for example, variables, function parameters and return values, and operators. The displayed

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

For more information, see [Interpret Results](#).

Identification of variables exceeding specified range limits

By default, Polyspace Code Prover performs a *robustness* verification of your code. The verification proves that the software works under all conditions. As the verification assumes that all data inputs are set to their full range, almost any operation on these inputs can produce an overflow.

To prove that your code works in normal conditions, use the Data Range Specification (DRS) feature to perform contextual verification. You can set constraints on data ranges, and verify your code within these ranges. The use of DRS can substantially reduce the number of orange checks in verification results.

You can use DRS to set constraints on:

- Global variables
- Input parameters for user-defined functions called by the main generator
- Return values for stub functions

For a global variable, if you specify the `globalassert` mode, the software generates a warning when the variable exceeds your specified range.

For more information, see [Data Range Configuration](#).

Quality metrics for tracking conformance to software quality objectives

You can define a quality model with reference to coding rule violations, code complexity, and run-time errors. By observing these metrics, you can track your progress toward predefined software quality objectives as your code evolves from the first iteration to the final version.

By confirming the absence of certain run-time errors and measuring the rate of improvement in code quality, Polyspace Code Prover enables developers, testers, and project managers to produce, assess, and deliver code that is free of run-time errors.

For more information, see [Quality Metrics](#).

Web-based dashboard providing code metrics and quality status

Polyspace Code Prover provides Polyspace Metrics, a Web-based dashboard for tracking submitted verification jobs, reviewing progress, and viewing the quality status of your code. Polyspace Metrics provides an integrated view of project metrics, displaying code complexity, coding rule violations, run-time errors, and other code metrics.

For more information, see [Quality Metrics](#).

Guided review-checking process for classifying results and run-time error status

In the Results Manager perspective, Polyspace Code Prover provides you with several options to organize your review process.

- You can use review methodologies to specify the number and type of checks displayed on the **Results Summary** pane. With each methodology, you review only a subset of checks.

For example, if you are reviewing verification results for the first time, select **First checks to review**. The software displays all red and gray checks but only a subset of orange checks. These orange checks are the ones most likely to be run-time errors. For more information, see [Review Checks Using Predefined Methodologies](#).

- You can group checks by **File/Function** or **Check**:
 - Grouping by **Check** classifies checks by color. Within each color, this grouping classifies checks by categories related to the origin of the check, such as **Control flow**, **Data flow**, and **Numerical**.
 - Grouping by **File/Function** classifies checks by the file where they originated. Within each file, this grouping classifies checks by functions where they originated.
 - For C++ files, you can also group checks by **Class**. This grouping classifies checks by the class definition where they originated.

For more information, see [Organize Check Review Using Filters and Groups](#).

- You can filter checks using any of the column information criteria on the **Results Summary** pane. For example, you can filter out checks that you have already justified using the filter icon on the **Justified** column header. If you have applied a filter, the column heading changes to indicate that all results are not displayed.

You can also define custom filters. For more information, see [Organize Check Review Using Filters and Groups](#).

- You can navigate through the **Results Summary** pane using the keyboard or UI buttons. Both means of navigation respect the grouping, filters, and methodology used to display results.

Graphical display of variable reads and writes

A Polyspace Code Prover verification generates a data dictionary with information about global variables and the read and write access operations on these variables. You can view this information through the **Variable Access** pane of the Results Manager perspective.

For more information, see [Exploring Results Manager Perspective](#).

Comparison with R2013a Polyspace products

Polyspace Code Prover is a single product that replaces the following R2013a products:

- Polyspace Client™ for C/C++
- Polyspace Server™ for C/C++

Polyspace Bug Finder™, which is available with the Polyspace Code Prover, incorporates the following R2013a products:

- Polyspace Model Link™ SL
- Polyspace Model Link TL
- Polyspace UML Link™ RH

For a summary of differences and similarities in remote verification, results review and other features and options, expand the following:

Remote verification

Category	R2013a	R2013b
Products required	Install: <ul style="list-style-type: none">• Polyspace Client for C/C++ on local computer	Install: <ul style="list-style-type: none">• MATLAB, Polyspace Bug Finder, and Parallel Computing Toolbox on local computer.

Category	R2013a	R2013b
	<ul style="list-style-type: none"> Polyspace Server for C/C++ on network computers, which are configured as Queue Manager and CPUs. 	<ul style="list-style-type: none"> MATLAB, Polyspace Bug Finder, Polyspace Code Prover, and MATLAB Distributed Computing Server on head node of computer cluster. For information about setting up a cluster, see Install Products and Choose Cluster Configuration.
Configuring and starting services	<p>On the Polyspace Preferences > Server Configuration tab:</p> <ul style="list-style-type: none"> Under Remote configuration, specify host computer for Queue Manager and Polyspace Metrics server and communication port. Under Metrics configuration, specify other settings for Polyspace Metrics. 	<p>On the Polyspace Preferences > Server Configuration tab:</p> <ul style="list-style-type: none"> Under MDCS cluster configuration, specify computer for cluster head node, which hosts the MATLAB job scheduler (MJS). The MJS replaces the R2013a Polyspace Queue Manager. Under Metrics configuration: <ul style="list-style-type: none"> Specify host computer for Polyspace Metrics server and communication port. Specify other settings for Polyspace Metrics.


Category	R2013a	R2013b
	<p>In the Remote Launcher Manager dialog box:</p> <ol style="list-style-type: none"> 1 Under Common Settings, specify Polyspace communication port, user details, and results folder for remote verifications. 2 Under Queue Manager Settings, specify Queue Manager and CPUs. 3 Under Polyspace Server Settings, specify available Polyspace products. 4 To start the Queue Manager and Polyspace Metrics service, click Start Daemon. 	<p>In the Metrics and Remote Server Settings dialog box:</p> <ol style="list-style-type: none"> 1 Under Polyspace Metrics Settings, specify user details, Polyspace communication port, and results folder for remote verifications. 2 Under Polyspace MDCS Cluster Security Settings, you see the following options with default values: <ul style="list-style-type: none"> • Start the Polyspace MDCE service — Selected. The <code>mdce</code> service, which is required to manage the MJS, runs on the MJS host computer and other nodes of the cluster. • MDCE service port — 27350. • Use secure communication – Not selected. Communication is not encrypted. You may want to use communication with security. For information about MATLAB Distributed Computing Server cluster security, see Cluster Security. 3 To start the Polyspace Metrics and <code>mdce</code> services, click Start Daemon. <p>Use the Metrics and Remote Server Settings dialog box to start and stop <code>mdce</code> services only if you configure the MDCS head node as the Polyspace Metrics server. Otherwise, clear the</p>

Category	R2013a	R2013b
		<p>Start the Polyspace MDCE service check box, and use the MDCS Admin Center. To open the MDCS Admin Center, run:</p> <pre><i>MATLAB_Install/toolbox/distcomp/bin/admincenter</i></pre> <p>For information about the MDCS Admin Center, see Cluster Processes and Profiles.</p>
Running a remote verification	<p>In the Project Manager perspective:</p> <ol style="list-style-type: none"> On the Configuration > Machine Configuration pane, select the following check boxes: <ul style="list-style-type: none"> Send to Polyspace Server Add to results repository — Allows viewing of results through Polyspace Metrics. On the toolbar, click Run. <p>The Polyspace client performs code compilation and coding rule checking on the local, host computer. Then the Polyspace client submits the verification to the Queue Manager on your network.</p>	<p>In the Project Manager perspective:</p> <ol style="list-style-type: none"> On the Configuration > Distributed Computing pane, select the Batch check box. By default, the software selects the Add to results repository, which enables the generation of Polyspace Metrics. On the toolbar, click Run. <p>The Polyspace Code Prover software performs code compilation and coding rule checking on the local, host computer. Then the Parallel Computing Toolbox client submits the verification job to the MJS of the MATLAB Distributed Computing Server cluster.</p>
Managing remote verifications	<p>Use the Queue Manager to monitor and manage submitted jobs from Polyspace clients.</p> <p>On the Web, you can monitor jobs through Polyspace Metrics. If you have installed Polyspace Server for C/C++ on your local computer, through Polyspace Metrics, you can open the Queue Manager .</p>	<p>Use the Queue Manager to monitor and manage jobs submitted through Parallel Computing Toolbox clients.</p>

Category	R2013a	R2013b
Accessing results of remote verifications	<p>When you run a verification on a Polyspace server, the Polyspace software automatically downloads the results to your local, client computer. You can view the results in the Results Manager perspective. In addition, you can use the Queue Manager to download results of verifications submitted from other Polyspace clients.</p> <p>On the Web, use Polyspace Metrics to view verification results stored in results repository. If Polyspace Client for C/C++ is installed on your local computer, you can download verification results. For example, in Polyspace Metrics, clicking a cell value in the Run-Time Checks view opens the corresponding verification results in the Results Manager.</p>	<p>On the Web, use Polyspace Metrics to view verification results. If Polyspace Bug Finder is installed on your local computer, you can download verification results. For example, in Polyspace Metrics, clicking a Project cell in the Runs view opens the corresponding verification results in the Results Manager.</p>

Results review

Category	R2013a	R2013b
Results Explorer	<p>Available. Allows navigation through checks by the file and function where they occur. To view, select Window > Show/Hide View > Results Explorer.</p>	<p>Removed. To navigate through checks by file and function, on Results Summary pane, from the drop-down menu, select File/Function.</p>
Filters on the Results Summary pane	<p>Filters appear as icons on the Results Summary pane. You can filter by:</p> <ul style="list-style-type: none"> • Run-time error category • Coding rules violated 	<p>You can filter by the information in all the columns of the Results Summary pane. In addition to existing filters, the new filtering capabilities extend to the file, function and line number where the checks</p>

Category	R2013a	R2013b
	<ul style="list-style-type: none"> • Check color • Check justification • Check classification • Check status 	<p>appear. You can also define your own filters.</p> <p>The filters appear as the  icon on each column header. To apply a filter using the information in a column:</p> <ol style="list-style-type: none"> 1 Place your cursor on the column header. The filter icon appears. 2 Click the filter icon and from the context menu, clear the All box. Select the appropriate boxes to see the corresponding checks. <p>For more information, see Organize Check Review Using Filters and Groups.</p>
Code Coverage Metrics	<p>In the Results Explorer view, the software displays two metrics for the project:</p> <ul style="list-style-type: none"> • unp — Number of unreachable functions as a ratio of total number of functions • cov — Percentage of elementary operations covered by verification <p>The unreachable procedures are marked gray in the Results Explorer view.</p>	<p>The new Results Statistics pane displays the code coverage metrics through the Code covered by verification column graph.</p> <p>To see a list of unreachable procedures, click this column graph.</p> <p>For more information, see Results Statistics.</p>

Other features

Product	Feature	R2013a	R2013b
Polyspace Client and Server for C/C++	Installation	Separate installation process for Polyspace products	Polyspace Code Prover software installed during MATLAB installation process.
	Project configuration	On host, for example, using Polyspace Client for C/C++ software.	On host, using Polyspace Code Prover software.
	Local verification	On host, run Polyspace Client for C/C++ verification. Review results in Results Manager.	On host, run Polyspace Code Prover verification. Review results in Results Manager.
	Export of review comments to Excel [®] , and Excel report generation	Supported	Not supported.
	Line command	<code>polyspace-c ...</code> <code>polyspace-cpp ...</code>	<code>polyspace-code-prover-nodesktop ...</code>
	Project configuration file extension	<code>project_name.cfg</code>	<code>project_name.psprj</code>
	Results file extension	<code>results_name.rte</code>	<code>results_name.pscp</code>
	Configuration > Machine Configuration pane	Available	Replaced by Configuration > Distributed Computing pane.
	Configuration > Post Verification pane	Available	Renamed Configuration > Advanced Settings
	<code>goto</code> blocks	Not supported	Supported

Product	Feature	R2013a	R2013b
	Run verifications from multiple Polyspace environments	Supported	Not supported, produces a license error - 4, 0.
	Non-official options field	Available in Configuration > Machine Configuration pane	Renamed Other and moved to Configuration > Advanced Settings pane
Polyspace Model Link SL and TL	Default includes	Includes specific to the target specified.	Generic includes for C and C++. These includes are target independent.
	Running a verification	Code > Polyspace > Polyspace for Embedded Coder/ Target Link <ul style="list-style-type: none"> • Verify Generated Code • Verify Generated Model Reference Code <p>Also right-clicking on a subsystem and selecting Polyspace > Polyspace for Embedded Coder/ Target Link</p>	Code > Polyspace > Verify Code Generated for <ul style="list-style-type: none"> • Selected Subsystem • Model • Referenced Model • Selected Target Link Subsystem <p>Also right-clicking on a subsystem and selecting Polyspace > Verify Code Generated for > Selected Subsystem / Selected Target Link Subsystem</p>
	Product Mode	Not available.	Choose between Code Prover or Bug Finder depending on the type of analysis you want to run.
	Settings	Available. Called Verification Settings from	Available. Called Settings from . Functionality the same.

Product	Feature	R2013a	R2013b
	Open results	Option Open Project Manager and Results Manager opened the Polyspace Project Manager.	Option Open results automatically after verification opens Polyspace Metrics (batch verifications) or Polyspace Results Manager (local verifications).
Polyspace plug-in for Visual Studio 2010	Support for C++11 features	Partial support.	Added support for: <ul style="list-style-type: none"> • Lambda functions • Rvalue references for <code>*this</code> and initialization of class objects by rvalues • Decltype • Auto keyword for multi-declarator auto and trailing return types • Static assert • Nullptr • Extended friend declarations • Local and unnamed types as template arguments

Options

Product	Option	R2013a	R2013b
	<code>-code-metrics</code>	Available. Not selected by default.	Removed. Code complexity metrics computed by default.

Product	Option	R2013a	R2013b
	-dialect	Available.	Default unchanged, but new value gnu4.6 available for C and C++.
Polyspace Client and Server for C/C++	-max-processes	Specify through Machine Configuration > Number of processes for multiple CPU core systems or command line .	Specify from command line, or through Advanced Settings > Other .
	-allow-language-extensions	Available. Selected by default.	Removed. By default, software supports subset of common C language constructs and extended keywords defined by the C99 standard or supported by many compilers.
	-enum-type-definition	Available with three values. First value called <code>defined-by-standard</code> .	Available with three values. For C, first value renamed <code>signed-int</code> . For C++, first value renamed <code>auto-signed-int-first</code> .

Product	Option	R2013a	R2013b
Polyspace Model Link SL and TL	- scalar - overflows - behavior wrap - around	Available. Not selected by default.	Default. This option identifies generated code from blocks with saturation enabled. However, this option might lead to a loss of precision. For models without saturation, you can choose to remove this option.
	- ignore - constant - overflows	Available. Not selected by default.	Default.

