Polyspace® Bug Finder™ Release Notes

MATLAB®&SIMULINK®

MathWorks®

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® Bug Finder™ Release Notes

Contents

R2014a

R2013b

R2015b

Version: 2.0

New Features

Bug Fixes

Compatibility Considerations

Results in Real Time: View results as they are produced

Previously, you could not review results until the analysis was complete. For local analyses in R2015b, you can start reviewing results as soon as they are available.

When you run a local analysis, a new button appears on the toolbar.
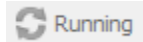
When results are available, this button becomes active.
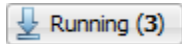
To start reviewing available results, click this button. The button reactivates every time results are available. To load additional results, click the button again.

When the analysis is complete, to load all your results, click the button.
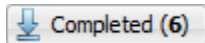
For more information, see "Open Results".

Mixed C/C++ Code: Run analysis on entire project with C and C++ source files

If your coding project contains C and C++ files, you can now analyze the entire project in one Polyspace® project. Use the new C/C++ setting to compile `.c` files with C compilation rules and compile `.cpp` and other files with C++ compilation rules.

To create a mixed C and C++ project:

- At the command line, use the option `-lang C-CPP`.
- In the user interface:

    1 Select **File** > **New Project**.

    2 In the Project properties window, select **Project Language** > **C++** as the main project language. Enter your other project properties as before.

    3 When adding source files, add your `.c` and `.cpp` files with their include files.

    4 In the configuration, on the **Target & Compiler** pane, set **Source code language** > **C-CPP**. This setting indicates to the compiler to use C compilation

rules for `.c` files and C++ compilation rules for `.cpp` files. For other file extensions, Polyspace uses C++ compilation rules.

5   Set your other options as required. Some limitations to consider:

   - Coding rules — You can select only one C coding rule set and one C++ coding rule set.
   - Bug Finder Defects — You can select C/C++ or C++ defects. The C++ defects are checked only on `.cpp` files.

For other changes to analysis options, see "Changes in analysis options" on page 1-21.

More Defect Categories: Detect security vulnerabilities, resource management issues, object oriented design issues

You can check your code against five new categories of defects:

- Resource management — Defects related to resource handling such as detection of unclosed file descriptors or use of a closed file descriptor.
- Object oriented — Defects related to C++ object-oriented programming such as detection of class design issues or issues in the inheritance hierarchy.
- Security — Defects related to security vulnerabilities such as vulnerable standard functions, use of sensitive data, and pseudo-random number generation.
- Tainted data — Defects related to using variables that someone outside your program can manipulate and externally controlled resources.
- Good practice — Defects that allow you to observe good coding practices such as detection of hard-coded memory buffer size or unused function parameters.

For information about the new defects, see "Changes to Bug Finder Defects" on page 1-15.

Autodetection of Multitasking Primitives: Analyze source code with multitasking primitives from POSIX and VxWorks without manual setup

If you use POSIX® or VxWorks® to perform multitasking, Polyspace can now interpret your multitasking code more easily.

Functions Polyspace can interpret:

POSIX

- `pthread_create`
- `pthread_mutex_lock`
- `pthread_mutex_unlock`

VxWorks

- `taskSpawn`
- `semTake`
- `semGive`

By default in R2015b, Polyspace detects thread creating and critical sections from supported multitasking functions.

For more information, see "Modeling Multitasking Code".

Improved Eclipse Support: View results embedded in source code and context-sensitive help

In R2015b, the following improvements have been made to the Polyspace plugin for Eclipse™:

- Polyspace Bug Finder™ highlights defects in your source code in the following ways:

  - For defects, an ! mark appears before the line number on the left. For coding rule violations, a ▽ or ▼ mark appears before the line number on the left.
  - The operation containing the defect has a wavy red underlining.
  - For defects, a ▭ icon appears in the overview ruler to the right of the line containing the defect. For coding rule violations, a ▭ icon appears in the overview ruler to the right of the line containing the rule violation. If you place your cursor on the icon, a tooltip shows a brief description of the defect or coding rule.

    In addition, a ■ icon appears at the top of the overview ruler. If you place your cursor on the icon, a tooltip states the total number of defects and coding rule violations in the file.

  Using these indicators, you can track defects in your source code more easily. For more information, see "Review and Fix Results".

- When you select a result in the **Results Summary - Bug Finder** view, the **Result Details** view displays additional information about the result. In the **Result**

Details view, if you click the ⓘ button next to the result name, you can see a brief description and examples of the result. For defects, you can sometimes see the risk associated with not fixing the defect and the most common fix for the defect.

- You can switch to a Polyspace perspective that shows only the information relevant to a Polyspace Bug Finder analysis. To open the perspective, select **Window** > **Open Perspective** > **Other**. In the Open Perspective dialog box, select **Polyspace**.

  Once you switch to the Polyspace perspective, the source code shows the Polyspace Bug Finder defects only in this perspective.

- You can view results as they are produced instead of waiting till end of the analysis.

  - When you begin an analysis, a 🔄 icon appears next to the ▶ button.

  - If results are available, the icon turns to ⬇. Click the ⬇ icon to load available results.

  - With your results open, if additional results are available, the ⬇ icon is still visible. Click the ⬇ icon to load all available results.

Complete MISRA C:2012 Support: Detect violations of all MISRA C:2012 rules

In R2015b, Polyspace Bug Finder supports the following MISRA C®: 2012 coding rules.

Rule	Description
MISRA C:2012 Directive 2.1	All source files shall compile without any compilation errors.
MISRA C:2012 Directive 4.5	Identifiers in the same name space with overlapping visibility should be typographically unambiguous.
MISRA C:2012 Directive 4.13	Functions which are designed to provide operations on a resource should be called in an appropriate sequence.
MISRA C:2012 Rule 2.6	A function should not contain unused label declarations.
MISRA C:2012 Rule 2.7	There should be no unused parameters in functions.
MISRA C:2012 Rule 17.5	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements.

Rule	Description
`MISRA C:2012 Rule 17.8`	A function parameter should not be modified.
`MISRA C:2012 Rule 21.12`	The exception handling features of `<fenv.h>` should not be used.
`MISRA C:2012 Rule 22.1`	All resources obtained dynamically by means of Standard Library functions shall be explicitly released.
`MISRA C:2012 Rule 22.2`	A block of memory shall only be freed if it was allocated by means of a Standard Library function.
`MISRA C:2012 Rule 22.3`	The same file shall not be open for read and write access at the same time on different streams.
`MISRA C:2012 Rule 22.4`	There shall be no attempt to write to a stream which has been opened as read-only.
`MISRA C:2012 Rule 22.5`	A pointer to a `FILE` object shall not be dereferenced.
`MISRA C:2012 Rule 22.6`	The value of a pointer to a `FILE` shall not be used after the associated stream has been closed.

Defects Classified by Impact: Prioritize defect review by using the impact attribute assigned to each defect type

You can prioritize your result review using an **Impact** attribute assigned to the defects. The attribute is assigned based on the following considerations:

- Criticality, or whether the defect is likely to cause a code failure.
- Certainty, or the rate of false positives.

You can filter results on the **Results Summary** pane using the **Impact** attribute. Or, you can obtain a graphical visualization of the **Defect distribution by impact** on the **Dashboard** pane. For more information, see "Classification of Defects by Impact".

Microsoft Visual C++ 2013: Analyze code developed in Microsoft Visual C++ 2013

You can analyze code developed in the Microsoft® Visual C++® 2013 dialect.

To analyze code compiled with Microsoft Visual C++ 2013, set your dialect to `visual12.0`. Once you specify your dialect, Microsoft Visual C++ allows language

extensions specific to Microsoft Visual C++ 2013. For more information, see "Dialect (C)" or "Dialect (C++)".

GNU 4.9 and Clang 3.5 Support: Analyze code compiled with GNU 4.9 or Clang 3.5

Polyspace now supports the GNU 4.9 and Clang 3.5 dialects for C and C++ projects.

To analyze code compiled with one of these dialects, set the **Target & Compiler** > **Dialect** option to gnu4.9 or clang3.5.

For more information, see "Dialect (C)" or "Dialect (C++)".

Improved Review Capability: View result details and add review comments in one window

In R2015b, the **Check Details** pane is renamed as **Result Details**. On this pane, you can now enter review information such as **Classification**, **Status**, and comments. For more information, see "Review and Fix Results".
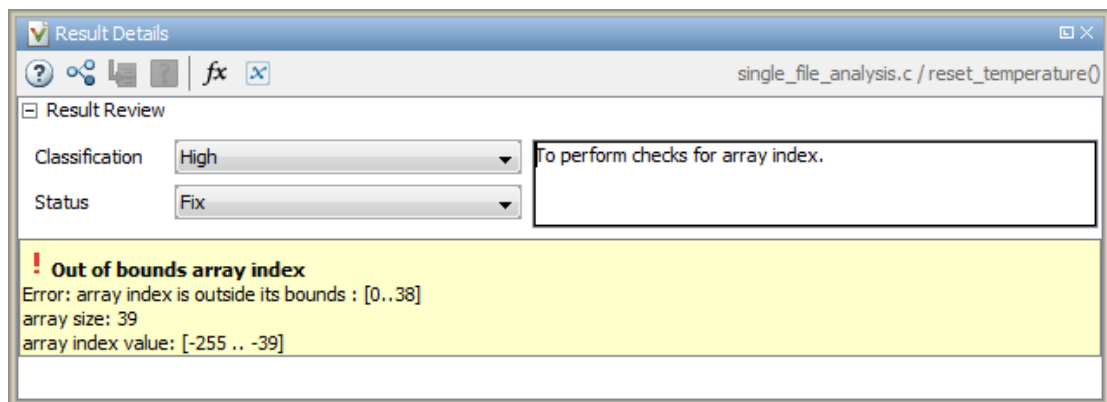
Previously, to enter review information while keeping the **Results Summary** pane collapsed, you used the **Check Review** pane. This pane has been removed.

Saved Layouts: Save your preferred layouts of the Polyspace user interface

In R2015b, if you reorganize the Polyspace user interface and place the various panes in more convenient locations, you can save your new layout. If you change your layout, you can quickly revert to a saved layout.

With this modification, you can create customized layouts suitable for different requirements. You can switch between saved layouts quickly. For instance:

- You can have separate layouts for project configuration and results review.
- You can have a minimal layout with only the frequently used panes.

For more information, see "Organize Layout of Polyspace User Interface".

Start Page: Get oriented with Polyspace Bug Finder

In R2015b, when you open Polyspace Bug Finder for the first time, a **Start Page** pane appears. From this pane, you can:

- Open Polyspace recent results and examples.
- Start a new project.
- Get additional help using the **Getting Started**, **What's New**, and **Learn More** tabs.

If you select the **Show on startup** box, the pane appears each time you open Polyspace Bug Finder. Otherwise, if you close the pane once, it does not reopen. To open the pane, select **Window > Show/Hide View > Start Page**.

Enhanced Review Scope: Filter coding rule violations from display in one click

Previously, using custom options on the **Show** menu, you suppressed only defects and code metrics (if they fell below a certain threshold). In R2015b, you can suppress a certain number or percentage of coding rule violations from the display. You use custom options in the **Show** menu on the **Results Summary** pane. You can:

- Suppress violations of coding rules that are not relevant.
- Focus your results review by seeing only a certain number of coding rule violations in your display.

• Predefine a percentage of coding rule violations that you intend to review and view only that percentage in your analysis results.

You define an option on the **Show** menu only once. The option is available for one-click use every time that you open your results. For information on how to create an option to suppress coding rule violations, see "Suppress Certain Rules from Display in One Click".

Improvements to automatic project creation from build command

In R2015b, automatic project creation from your build command is improved:

• If you build your source code from the Cygwin™ environment (using either a 32-bit or 64-bit installation), Polyspace can trace your build and to create a Polyspace project or options file.

• Support for the following compilers has improved:

   • Texas Instruments™ C2000 compiler

      This compiler is available with Code Composer Studio™.

   • Cosmic HC08 C compiler

   • MPLAB XC8 C Compiler

• With certain compilers, the speed of tracing your build command has improved. The software now stores build information in the system temporary folder, thereby allowing faster access during the build.

   If you still encounter a slow build, use the advanced option `-cache-path ./ ps_cache` when tracing your build. For more information, see "Slow Build Process When Polyspace Traces the Build".

• If the software detects target settings that correspond to a standard processor type, it assigns that standard target processor type to your project. The target processor type defines the size of fundamental data types and the endianness of the target machine. For more information, see "Target processor type (C/C++)".

   Previously, when you created a project from your build command, the software assigned a custom target processor type. Although you saw the processor type in the form of an option such as `-custom-target true,8,2,4,-1,4,8,4,8,8,4,8,1,little,unsigned_int,int,unsigned_short`, you could not identify easily how many bits were associated with each fundamental type. With this enhancement, when the software assigns a processor type, you can

identify the number of bits for each type. Click the **Edit** button for the option **Target processor type**.

- Automatic project creation uses a configuration file written for specific compilers. If your compiler is not supported, you can adapt one of the existing configuration files for your compiler. The configuration file, written in XML, is now simplified with some new elements, macros and attributes.

  - The `preprocess_options_list` element supports a new `$(OUTPUT_FILE)` macro when the compiler does not allow sending the preprocessed file to the standard output.

  - A new `preprocessed_output_file` element allows the preprocessed file name to be adapted from the source file name.

  - The `semantic_options` element supports a new `isPrefix` attribute. This attribute provides a shortcut to specify multiple semantic options that begin with the same prefix.

  - The `semantic_options` element supports a new `numArgs` attribute. This attribute provides a shortcut to specify semantic options that take one or more arguments.

  For more information, see "Compiler Not Supported for Project Creation from Build Systems".

- Sometimes, the build command returns a non-zero status even when the command succeeds. The non-zero status can result from warnings in the build process. However, Polyspace does not trace the build and create a Polyspace project. You can now use an option `-allow-build-error` to create a Polyspace project even if the build command returns an exit status or error level different from zero. This option helps you understand the error in the build process.

  For more information, see `-option value` arguments of `polyspaceConfigure`.

Improvements in checking of previously supported MISRA C rules

In R2015b, the following changes have been made in MISRA C checking:

MISRA C:2004

Rule	Description	Improvement
`MISRA C:2004 Rule 2.1`	Assembly language shall be encapsulated and isolated.	If an assembly language statement is entirely encapsulated in macros, Polyspace no longer considers that a statement a violation of this rule
`MISRA C:2004 Rule 8.8`	An external object or function shall be declared in one file and only one file.	Polyspace considers that variables or functions declared `extern` in a non-header file violate this rule.
`MISRA C:2004 Rule 10.1`	The value of an expression of integer type shall not be implicitly converted to a different underlying type if it is not a conversion to a wider integer type of the same signedness.	Polyspace no longer raises violation of this rule on operations involving pointers.
`MISRA C:2004 Rule 19.2`	Nonstandard characters should not occur in header file names in `#include` directives.	If the character \ or \\ occurs between the < and > in `#include <filename>` (or between " and " in `#include "filename"`), Polyspace no longer raises violation of this rule. Therefore, you can use Windows® paths to files in place of `filename` without triggering a rule violation.

MISRA C:2012

Rule	Description	Improvement
`MISRA C:2012 Directive 4.3`	Assembly language shall be encapsulated and isolated.	If an assembly language statement is entirely encapsulated in macros, Polyspace no longer considers that a statement a violation of this rule

Rule	Description	Improvement
`MISRA C:2012 Rule 1.1`	The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits.	If your `.c` file contains too many macros, Polyspace places the rule violation at the beginning of the file instead on the last macro usage. Therefore, you can add a comment before the first line of the `.c` file justifying the violation. Previously, if you placed a justification comment before the last macro usage and later added another macro usage, the comment no longer applied. For information on adding code comments to justify results, see "Annotate Code for Rule Violations".
`MISRA C:2012 Rule 10.4`	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.	• If one of the operands is the constant zero, Polyspace does not raise a violation of this rule. • If one of the operands is a signed constant and the other operand is unsigned, the rule violation is not raised if the signed constant has the same representation as its unsigned equivalent. For instance, the statement `u8b = u8a + 3;`, where `u8a` and `u8b` are `unsigned char` variables, does not violate the rule because the constants `3` and `3U` have the same representation.

Checking Coding Rules Using Text Files

In R2015b, if your coding rules configuration text file has an incorrect syntax, the analysis stops with an error message. The error message states the line numbers in the configuration file that contain the incorrect syntax.

For more information on checking for coding rules using text files, see "Format of Custom Coding Rules File".

Including options multiple times

You can specify analysis options multiple times. This new capacity is available only at the command line or using the command-line names in the **Advanced options** pane in the user interface. You can customize pre-made configurations without having to remove options.

If you specify an option multiple times, only the last setting is used. For example, if your configuration is:

```
-lang c
-prog test_bf_cp
-verif-version 1.0
-author username
-sources-list-file sources.txt
-OS-target no-predefined-OS
-target i386
-dialect none
-misra-cpp required-rules
-target powerpc
```
Polyspace uses the last target setting, `powerpc`, and ignores the other target specified, `i386`.

In the user interface, if you specify **c18** as the target on the Target and Compiler pane and in **Advanced options** enter `-target i386`, these two targets count as multiple analysis option specifications. Polyspace uses the target specified in the Advanced options dialog box, `i386`.

Renaming of labels in Polyspace user interface

In the Polyspace user interface, the following labels have been renamed:

- On the **Configuration** pane, the **Coding Rules** node is renamed **Coding Rules & Code Metrics**.

  The new **Coding Rules & Code Metrics** node now contains the option **Calculate Code Metrics**, which previously appeared in the **Advanced Settings** node.

- On the **Results Summary** pane, the **Category** column title is changed to **Group**. This change avoids confusion with coding rule categories.

- On the **Results Summary** and **Result Details** pane, the field **Classification** is changed to **Severity**. You assign a **Severity** such as High, Medium and Low to a defect to indicate how critical you consider the issue.

- The labels associated with specifying constraints have changed as follows:

  - On the **Configuration** pane, the field **Variable/function range setup** is changed to **Constraint setup**.

  - When you click **Edit** beside the Constraint Setup field, a new window opens. The window name is changed from **Polyspace DRS Configuration** to **Constraint Specification**.

  For more information, see "Specify Constraints".

Configuration Associated with Result Not Opened by Default

In R2015b, when you open your result, the **Configuration** pane does not automatically display a read-only form of the associated configuration.

To view the configuration associated with the result, select the link **View configuration for results** on the **Dashboard** pane. If a corresponding project is open in the **Project Browser**, you can also right-click the **Results** node in the project and select **Open Configuration**.

Improvements in Report Templates

In R2015b, the major improvements in report templates include the following:

- The summary chapter in the template **BugFinder** now contains a breakup of Polyspace Bug Finder results by file, in addition to the project-wide summary.

- The summary now shows the total number of results along with the number of results reviewed.

- Instead of filenames, absolute paths to files appear in the reports.

- If you check for coding rules, the appendix about coding rules configuration states all rules along with the information whether they were enabled or disabled. Previously, the appendix only stated the enabled rules.

- The reports display the impact attribute associated with a defect.

  For more information on this attribute, see "Classification of Defects by Impact".

For more information on templates, see "Report template (C/C++)".

Updated Support for TargetLink

The Polyspace plug-in for TargetLink® now supports versions 3.5 and 4.0 of the dSPACE® Data Dictionary and TargetLink Code Generator.

dSPACE and TargetLink version 3.4 is no longer supported.

For more information, see "TargetLink Considerations".

Changes to Bug Finder Defects

- "New Defects" on page 1-15
- "Updated Defects" on page 1-21

The following tables list updates and additions to the list of Bug Finder defect checkers.

New Defects

Tainted Data Defects

Name	Description
Array access with tainted index	Array index from unsecure source possibly outside array bounds
Command executed from externally controlled path	Path argument from an unsecure source
Execution of externally controlled command	Command argument from an unsecure source is vulnerable to OS command injection
Host change using externally controlled elements	Changing host id from an unsecure source
Library loaded from externally controlled path	Library argument from an externally controlled path

Name	Description
Loop bounded with tainted value	Loop controlled by a value from an unsecure source
Memory allocation with tainted size	Size argument to memory function is from an unsecure source
Pointer dereference with tainted offset	Offset is from an unsecure source and dereference may be out of bounds
Tainted division operand	Division operands from an unsecure source
Tainted modulo operand	Remainder operands from an unsecure source
Tainted NULL or non-null-terminated string	Argument is from an unsecure source and may be NULL or not NULL-terminated
Tainted sign change conversion	Value from an unsecure source changes sign
Tainted size of variable length array	Size of the variable-length array (VLA) is from an unsecure source and may be zero, negative, or too large
Tainted string format	Input format argument is from an unsecure source
Use of externally controlled environment variable	Value of environment variable from an unsecure source
Use of tainted pointer	Pointer from an unsecure source may be NULL or point to unknown memory

Good Practice Defects

Name	Description
Delete of void pointer	delete operates on a void* pointer pointing to an object
Hard coded buffer size	Size of memory buffer is a numerical value instead of symbolic constant
Hard coded loop boundary	Loop boundary is a numerical value instead of symbolic constant
Unused parameter	Function prototype has parameters not read or written in function body
Use of setjmp/longjmp	setjmp and longjmp cause deviation from normal control flow

Programming Defects

Name	Description
Bad file access mode or status	Access mode argument of function in `fopen` or `open` group is invalid
Call to memset with unintended value	`memset` or `wmemset` used with possibly incorrect arguments
Copy of overlapping memory	Source and destination arguments of a copy function have overlapping memory
Exception caught by value	`catch` statement accepts an object by value
Exception handler hidden by previous handler	`catch` statement is not reached because of an earlier `catch` statement for the same exception
Improper array initialization	Incorrect array initialization when using initializers
Incorrect pointer scaling	Implicit scaling in pointer arithmetic might be ignored
Invalid assumptions about memory organization	Address is computed by adding or subtracting from address of a variable
Invalid va_list argument	Variable argument list used after invalidation with `va_end` or not initialized with `va_start` or `va_copy`
Modification of internal buffer returned from nonreentrant standard function	Function attempts to modify internal buffer returned from a nonreentrant standard function
Overlapping assignment	Memory overlap between left and right sides of an assignment
Possible misuse of sizeof	Use of `sizeof` operator can cause unintended results
Possibly unintended evaluation of expression because of operator precedence rules	Operator precedence rules cause unexpected evaluation order in arithmetic expression
Standard function call with incorrect arguments	Argument to a standard function does not meet requirements for use in the function
Use of memset with size argument zero	Size argument of function in `memset` family is zero

Name	Description
Variable length array with nonpositive size	Size of variable-length array is zero or negative
Writing to const qualified object	Object declared with a `const` qualifier is modified

Resource Management Defects

Name	Description
Closing a previously closed resource	Function closes a previously closed stream
Resource leak	File stream not closed before `FILE` pointer scope ends or pointer is reassigned
Use of previously closed resource	Function operates on a previously closed stream
Writing to read-only resource	File opened earlier as read-only is modified

Security Defects

Name	Description
Deterministic random output from constant seed	Seeding routine uses a constant seed making the output deterministic
Execution of a binary from a relative path can be controlled by an external actor	Command with relative path is vulnerable to malicious attack
File access between time of check and use (TOCTOU)	File/directory may have changed state due to access race
File manipulation after chroot() without chdir("/")	Path-related vulnerabilities for file manipulated after call to `chroot`
Function pointer assigned with absolute address	Constant expression is used as function address is vulnerable to code injection
Incorrect order of network connection operations	Socket is not correctly established due to bad order of connection steps or missing steps

Name	Description
Load of library from a relative path can be controlled by an external actor	Library loaded with relative path is vulnerable to malicious attacks
Mismatch between data length and size	Data size argument is not computed from actual data length
Missing case for switch condition	Default case is missing and may be reached
Predictable random output from predictable seed	Seeding routine uses a predictable seed making the output predictable
Sensitive data printed out	Function prints out sensitive data
Sensitive heap memory not cleared before release	Sensitive data not cleared or released by memory routine
Umask used with chmod-style arguments	Unsafe argument to `umask` allows external user too much control
Uncleared sensitive data in stack	Variable in stack is not cleared and contains sensitive data
Unsafe standard encryption function	Function is not reentrant or uses a risky encryption algorithm
Unsafe standard function	Function unsafe for security-related purposes
Use of dangerous standard function	Dangerous functions cause possible buffer overflow in destination buffer
Vulnerable path manipulation	Path argument with `/../`, `/abs/path/`, or other unsecure elements
Vulnerable permission assignments	Argument gives read/write/search permissions to external users
Vulnerable pseudo-random number generator	Using a cryptographically weak pseudo-random number generator
Use of non-secure temporary file	Temporary generated file name is unsecure

Name	Description
Use of obsolete standard function	Obsolete routines can cause security vulnerabilities and/or portability issues

Object-Oriented Defects

Name	Description
`*this not returned in copy assignment operator`	`operator=` method does not return a pointer to the current object
Base class assignment operator not called	Copy assignment operator does not call copy assignment operators of base subobjects
Base class destructor not virtual	Class cannot behave polymorphically for deletion of derived class objects
Copy constructor not called in initialization list	Copy constructor does not call copy constructors of some members or base classes
Incompatible types prevent overriding	Derived class method hides a `virtual` base class method instead of overriding it
Missing explicit keyword	Constructor missing the `explicit` specifier
Missing virtual inheritance	A base class is inherited both virtually and non-virtually in the same hierarchy
Member not initialized in constructor	Constructor does not initialize some members of a class
Object slicing	Derived class object passed by value to function with base class parameter
Partial override of overloaded virtual functions	Class overrides a fraction of the inherited virtual functions with a given name
Return of non const handle to encapsulated data member	Method returns pointer or reference to internal member of object
Self assignment not tested in operator	Copy assignment operator does not test for self-assignment

Updated Defects

Name	Status	Additional Information
• Integer conversion overflow • Integer overflow • Invalid use of standard library routine • Shift operation overflow • Sign change integer conversion overflow • Shift of a negative value • Unsigned integer conversion overflow • Unsigned integer overflow	Updated	The defects do not appear on computations involving constants only. For instance, the assignment `unsigned int var = -1;` does not show a Sign change integer conversion overflow defect.
Format string specifiers and arguments mismatch	Recategoriz	Moved from **Other** to **Programming**
Invalid use of standard library routine	Recategoriz	Moved from **Other** to **Programming**
Assertion	Recategoriz	Moved from **Other** to **Good practice**
Large pass-by-value argument	Recategoriz	Moved from **Other** to **Good practice**
Line with more than one statement	Recategoriz	Moved from **Other** to **Good practice**

Changes in analysis options

In R2015b, the following options have been added, changed, or removed.

New Options

Option	Status	Description
Respect C90 Standard	New	The analysis does not allow C language extensions that do not follow the ISO/IEC 9899:1990 standard.

Option	Status	Description
(-no-language-extensions)		
Dialect visual12.0	New	Allows Microsoft Visual C++ 2013 (visual 12) language extensions.
Dialect gnu4.9	New	Allows GCC 4.9 language extensions.
Dialect clang3.5	New	Allows Clang 3.5 language extensions.
"Source code language (C++)" (-lang)	New in the user interface	The -lang option is now available in the Polyspace user interface. It is on the **Target & compiler** tab and called **Source code language**.
"Source code language (C++)" > **C-CPP** (-lang C-CPP)	New option setting	For C++ projects, you can choose C-CPP to analyze a mix of .c and .cpp source files.
"Configure multitasking manually (C/C++)"	New	A user interface option only. This option enables the previous multitasking options • **Entry points** • **Critical section details** • **Temporally exclusive tasks**
"Disable automatic concurrency detection (C/C++)"	New	By default, the new automatic concurrency detection is enabled. If you want to turn it off, select this option.

Updated Options

Option	Change	Description
"Calculate Code Metrics (C/C++)"	Moved in user interface	The option has been moved in the Configuration panel from the **Advanced Settings** pane to the **Coding Rules and Code Metrics** pane.
"Signed right shift (C/C++)" (-logical-signed-right-shift)	Now available in C++ projects	

Option	Change	Description
"Division round down (C/C++)" (-div-round-down)	Now available in C++ projects	
Targets: • tms320c3x • , sharc21x61 • necv850 • hc08 • hc12 • mpc5xx • c18	Now available in C++ projects	
"Enum type definition (C/C++)" (-enum-type-definition)	Possible values updated	The possible values for -enum-type-definition now match for C and C++. Available values: • defined-by-standard (default) • auto-signed-first • auto-unsigned-first
-support-FX-option-results	No longer available in the user interface	
-pointer-is-24bits	Available in C++ projects	Available only if you use the **Target** setting c18.
-asm-begin -asm-end	Now available in C++ projects	
"Check MISRA C:2004"	Now available in C++ projects	Available only if you select **Source code language** > **C-CPP**.
"Check MISRA AC AGC"	Now available in C++ projects	Available only if you select **Source code language** > **C-CPP**.

Option	Change	Description
"Check MISRA C:2012" and "Use generated code requirements (C)"	Now available in C++ projects	Available only if you select **Source code language** > **C-CPP**.
"Effective boolean types (C)"	Now available in C++ projects	Available only if you select **Source code language** > **C-CPP**.
"Allowed pragmas (C)"	Now available in C++ projects	Available only if you select **Source code language** > **C-CPP**.
"Output format (C/C++)" `-report-output-format`	Possible values updated	The output format RTF is deprecated and not available on the **Configuration** pane.

Removed Options

Option	Status	Description
`-dialect cfront2`	Removed	Choose a different dialect.
`-dialect cfront3`	Removed	Choose a different dialect.
`-passes-time`	Removed	Polyspace includes this behavior by default. Remove this option from existing configurations.
`-include-headers-once`	Removed	Polyspace includes this behavior by default. Remove this option from existing configurations.
`-discard-asm`	Removed	This option is no longer supported. Remove this option from existing configurations.
`-misra2 AC-AGC-OBL-subset`	Removed	Use `-misra-ac-agc OBL-rules` instead.

Compatibility Considerations

If you use scripts that contain the removed or updated options, change your scripts accordingly.

Binaries removed

The following binaries have been removed.

Removed binary	Use instead
`polyspace-rl-manager.exe`	`polyspace-server-settings.exe`
`polyspace-spooler.exe`	`polyspace-job-monitor.exe`
`polyspace-ver.exe`	`polyspace-bug-finder-nodesktop -ver`

The binaries to use instead are located in *matlabroot*/`polyspace`/`bin`.

Support for Visual Studio 2008 to be removed

The Polyspace Add-In for Visual Studio® 2008 is no longer supported and will be removed in a future release.

Compatibility Considerations

To analyze your Visual Studio projects, use either:

- The Polyspace Add-in for Visual Studio 2010. See "Install Polyspace Add-In for Visual Studio".
- The `polyspace-configure` tool to create a project using your build command. See "Create Project Using Visual Studio Information".

Import Visual Studio project removed

The **Tools** > **Import Visual Studio project** has been removed.

To import your project information from Visual Studio, use the **Create from build system** option during new project creation. For more information, see "Create Project Using Visual Studio Information".

R2015a

Version: 1.3

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 have been unified. You can run the analysis and review results without switching between two perspectives.

The unification has resulted in the following major changes:

- After an analysis, the result opens automatically.

  Previously, after an analysis, you had to double-click the result in the **Project Browser** to open your new results.

- 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
• **Project Browser**: Set up project. • **Configuration**: Specify analysis options for your project. • **Output Summary**: Monitor progress of analysis. • **Run Log**: Find information about an analysis.	• **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. • **Results Properties**: Same as **Run Log**, but associated with results instead of a project. This pane has been removed. To open the log associated with a result, with the results open, select **Window** > **Show/Hide View** > **Run Log**. • **Settings**: Same information as **Configuration**, but associated with results instead of a project. This pane has been removed.

Project Manager	Results Manager
	To open the configuration associated with a result, with the results open, select **Window** > **Show/Hide View** > **Configuration**.

Code complexity metrics available in user interface

In R2015a, code complexity metrics can be viewed 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 exceeds this limit, the metric appears red in **Results Summary**.
- Comment and justify the value of a metric. If a metric value exceeds specified limits and appears red, you can add a comment with the rationale.

Using Polyspace results in this way, 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 analysis.

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 metrics results, 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

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 analysis, the new results are displayed with an asterisk (*) against them on the **Results Summary** pane. To display only these results, select the **New results** box.

If you make changes in your source code, you can use this feature to see only the results introduced due to those changes. You can avoid reviewing the results in your existing source code.

Search improvements 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 previously available search options.

Option to specify program termination functions

In R2015a, you can specify functions that behave like the exit function and terminate your program.

- At the command line, use the flag `-termination-functions`.
- In the user interface, on the **Configuration** pane, select **Advanced Settings**. Enter `-termination-functions` in the **Other** field.

For more information, see -termination-functions.

Simplified results infrastructure

Polyspace results folders are reorganized and simplified. Files have been removed, combined, renamed, or moved. The infrastructure changes do not change the analysis 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.psbf`. 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.

Default statuses to justify results

Polyspace Bug Finder results use certain statuses to calculate the number of justified results in Polyspace Metrics.

In R2015a, the default statuses that mark results as justified are:

- `Justified` — Previously called `Justify`, renamed in R2015a.
- `No action planned` — Existing status added to justified list in R2015a.

You can change which statuses mark results as justified from the Polyspace preferences. For more information, see Define Custom Review Status.

Filters to limit display of results

In R2015a, you can use the **Show** menu on the **Results Summary** pane to suppress certain Polyspace Bug Finder results from display.

- To suppress code complexity metrics from display, select **Show** > **Defects & Rules**.
- Create your own options on the **Show** menu. Select **Tools** > **Preferences** and create new options through the **Review Scope** tab.

  For more information, see Limit Display of Defects.

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 Bug Finder analysis, set the **Target & Compiler** > **Dialect** option to `gnu4.8`.

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

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 Number	Effect	More Information
Rule 12.1	More results on expressions with `sizeof` operator and on expressions with `?` 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 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 `try` block.	MISRA C++ Rules — Chapter 15

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.

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 model and highlights the related block.

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

For more information about the back-to-model feature, see Review Generated Code Results.

Changes to Bug Finder defects

Defect	R2015a change
Invalid use of floating point operation	Off by default.
Line with more than one statement	Off by default.
Invalid use of = (assignment) operator	On by default for handwritten code (analyses started at the command-line or Polyspace environment). Off by default for generated code (analyses started from the Simulink plug-in).
Invalid use of == (equality) operator	On by default for handwritten code. Off by default for generated code.
Missing null in string array	On by default for handwritten code. Off by default for generated code.
Partially accessed array	On by default for handwritten code. Off by default for generated code.

Defect	R2015a change
Variable shadowing	On by default for handwritten code. Off by default for generated code.
Write without further read	On by default for handwritten code. Off by default for generated code.
Wrong type used in sizeof	On by default for handwritten code. Off by default for generated code.

Polyspace binaries being removed

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

Binary name	Use instead
`polyspace-rl-manager.exe`	`polyspace-server-settings.exe`
`polyspace-spooler.exe`	`polyspace-job-monitor.exe`
`polyspace-ver.exe`	`polyspace-bug-finder-nodesktop -ver`

Import Visual Studio project being removed

The **Tools** > **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 Create Project Automatically.

R2014b

Version: 1.2

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.

Parallel compilation for faster analysis

Starting in R2014b, Polyspace Bug Finder can run the compilation phase of your analysis 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 analysis process.

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

Additional concurrency issue detection (deadlocks, double locks, and others)

Data race errors

The following defects deal with unprotected access of shared variables by multiple tasks.

Defect name	Status	More information
Race conditions	Removed	Replaced by Data race and Data race including atomic operations.

Defect name	Status	More information
Data race	New	Checks for unprotected operations on variables shared by multiple tasks. This check applies to non-atomic operations only.
Data race including atomic operations	New	Checks for unprotected operations on variables shared by multiple tasks. This check applies to all operations, including atomic ones.

Locking errors

The following defects deal with incorrect design of critical sections. For multitasking analysis, to mark a section of code as a critical section, you must place it between two function calls. A lock function begins a critical section. An unlock function ends a critical section.

Defect name	Status	More information
Deadlock	New	Checks whether the sequence of calls to lock functions is such that two tasks block each other.
Missing lock	New	Checks whether an unlock function has a corresponding lock function.
Missing unlock	New	Checks whether a lock function has a corresponding unlock function.
Double lock	New	Checks whether a lock function is called twice in a task without an unlock function being called in between.
Double unlock	New	Checks whether an unlock function is called twice in a task without a lock function being called in between.

For more information, see:

- Set Up Multitasking Analysis
- Review Concurrency Defects

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.

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 analysis options and defects

Contextual help is available for analysis options in the Polyspace interface and its plug-ins. To view the contextual help for analysis options:

1 Hover your cursor over an analysis option in the **Configuration** pane.

2 Inside the tooltip, select the "More Help" link.

   The documentation for that analysis option appears in a dockable window.

Contextual help is available for defects in the Polyspace interface. To view the contextual help:

1 In the Results Manager perspective, select a defect from the Results Summary.

2 Inside the **Check Details** pane, select ⓘ.

   The documentation for that Bug Finder defect appears in a dockable window.

For more information, see Getting Help.

Code editor in Polyspace interface

In R2014b, 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.

New and updated defect checkers

Defect name	Status	More information
Dead code	Updated	Checks for non-executed code. No longer checks for: • `if` conditions that are always true without a corresponding `else`. This check is covered by the Useless if defect. • Code following control-flow statements such as `break`, `return`, or `goto` defect. This check is covered by the Unreachable code defect.
Useless if	New	Checks for if-conditions that are always true.
Unreachable code	New	Checks for code following control-flow statements such as `break`, `return`, or `goto`.
Declaration mismatch	Updated	Updated for `#pragma` packing statements.
`Race conditions`	Removed	Replaced by Data race and Data race including atomic operations.
Data race	New	Checks for unprotected operations on variables shared by multiple tasks. This check applies to non-atomic operations only.
Data race including atomic operations	New	Checks for unprotected operations on variables shared by multiple tasks. This check applies to all accesses, including atomic ones.

Defect name	Status	More information
Deadlock	New	Checks whether the sequence of calls to lock functions is such that two tasks block each other.
Missing lock	New	Checks whether an unlock function has a corresponding lock function.
Missing unlock	New	Checks whether a lock function has a corresponding unlock function.
Double lock	New	Checks whether a lock function is called twice in a task without an unlock function being called in between.
Double unlock	New	Checks whether an unlock function is called twice in a task without a lock function being called in between.

Ignore files and folders during analysis

You can now use the analysis option **Files and folders to ignore** (command line `-includes-to-ignore`) to ignore files and folders during defect checking. Previously, the **Files and folders to ignore** option (command line `-includes-to-ignore`) ignored files and folders during coding rule checking. In R2014b, Polyspace Bug Finder uses this option to ignore specified files or folders for coding rule checking AND defect analysis.

For more information, see Files and folders to ignore (C) or Files and folders to ignore (C ++).

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 analysis 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 no longer 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, Polyspace can analyze this generated code. Polyspace uses the same default analysis options and parameters as Embedded Coder.

For more information, see Embedded Coder Considerations.

Remote launcher and queue manager renamed

Polyspace renamed the remote launcher and the queue manager.

Previous name	New name	More information
`polyspace-rl-manager`	`polyspace-server-settings`	Only the binary name has changed. The interface title, **Metrics and Remote Server Settings**, is unchanged.
`polyspace-spooler` **Queue Manager** or **Spooler**	`polyspace-job-monitor` **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.

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 analysis.	**Run > Run Report > Run Report**	**Reporting > Run Report**
Open a generated report.	**Run > Run Report > Open Report**	**Reporting > Open Report**
Import review comments from a previous analysis.	**Review > Import**	**Tools > Import Comments**
Specify code generator for generated code.	**Review > Code Generator Support**	**Tools > Code Generator Support**
Specify settings that apply to every Polyspace project.	**Options > Preferences**	**Tools > Preferences**
Specify settings for remote analysis.	**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 an analysis, you can view its progress on the **Output Summary** tab.
- In the **Project Browser**, projects appear sorted in alphabetical order instead of order of creation.

- On the **Configuration** pane, the **Interactive** option has been removed from the graphical interface. To use the interactive mode, use the `-interactive` flag at the command line, or in the **Advanced Settings** > **Other** text field. For more information, see -interactive

Improved Results Manager perspective

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

- To group your defects, use the **Group by** menu on the **Results Summary** pane.
  - To leave your defects ungrouped, instead of **List of Checks**, select **Group by** > **None**.
  - To group defects by category, instead of **Checks by Family**, select **Group by** > **Family**.
  - To group defects by file and function, instead of **Checks by File/Function**, select **Group by** > **File**.
- On the **Source** pane:
  - If a color appears on a brace enclosing a code block, double-click the brace to highlight the block. If no color appears, click the brace once to highlight the code block.
  - If a code block is deactivated due to conditional compilation, it appears gray.

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 an analysis.

Compatibility Considerations

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

Polyspace binaries being removed

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

Binary name	What happens	Use instead
`polyspace-rl-manager.exe`	Warning	`polyspace-server-settings.exe`
`polyspace-spooler.exe`	Warning	`polyspace-job-monitor.exe`
`polyspace-ver.exe`	Warning	`polyspace-bug-finder-nodesktop -ver`
`setup-remote-launcher.exe`	Warning	*MATLAB install*`/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 Create Projects Automatically from Your Build System.

R2014a

Version: 1.1

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:

- At the 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 folder from which you run your build command.
    - Additional options. For more information, see Create Project in User Interface.

    Click ▷ Run . 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.

Classification of bugs according to the Common Weakness Enumeration (CWE) standard

In R2014a, Polyspace Bug Finder associates CWE™ IDs with many defects. For the covered defects, the IDs are listed in the **CWE ID** column on the **Results Summary** pane. To view the **CWE ID** column, right-click the **Results Summary** tab and select the **CWE ID** column.

For more information, see Common Weakness Enumeration from Bug Finder Defects.

Additional coding rules support (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.

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, see Dialects for C or Dialects for C++.

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	R2013b	R2014a

New file wizard for custom coding rules.	For each coding rule, you can select three results:
	
	• **Error**: Analysis stops if the rule is violated.
	
	  The rule violation is displayed on the **Output Summary** tab in the Project Manager perspective.
	• **Warning**: Analysis continues even if the rule is violated.
	
	  The rule violation is displayed on the **Results Summary** pane in the Result Manager perspective.
	• **Off**: Polyspace does not check for violation of the rule.
		For each coding rule, you can select two results:
		
		• **On**: Analysis continues even if the rule is violated.
		
		  The rule violation is displayed on the **Results Summary** pane in the Result Manager perspective.
		• **Off**: Polyspace does not check for violation of the rule.

Format of the custom coding rules file.	Each line in the file must have the syntax:
	
	rule off|error|warning *#comments*
	
	For example:
	
	`# MISRA configuration - Proj1`
	`10.5 off #don't check 10.5`
	`17.2 error`
	`17.3 warning`
		Each line in the file must have the syntax:
		
		rule off|warning *#comments*
		
		For example:
		
		`# MISRA configuration - Proj1`
		`10.5 off #don't check 10.5`
		`17.2 warning`
		`17.3 warning`

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 analysis 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 analysis will stop if the rule is violated.

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.

Security level support for batch analysis

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 Set MJS Cluster Security.

Interactive mode for remote analysis

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

- 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-bug-finder-nodesktop` command.

- On the MATLAB command line, add the argument `'-interactive'` to the polyspaceBugFinder 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.
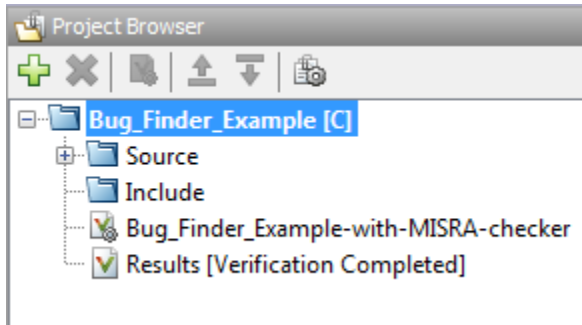
Results folder appearance in Project Browser

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
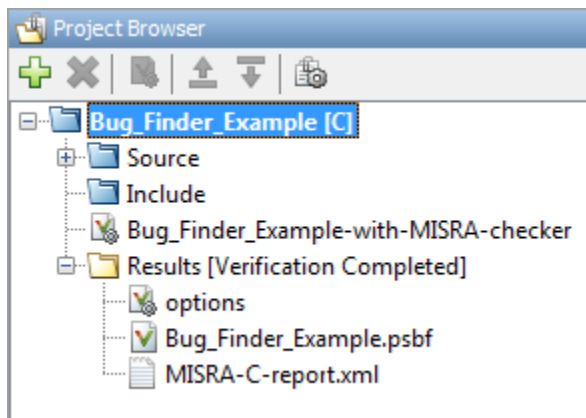
- Format in R2014a

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

Action	R2013b	R2014a
Open results.	In the result folder, double-click result file with extension `.psbf`.	Double-click result file.
Open analysis options used for result.	In the result folder, select **options**.	Right-click result file and select **Open Configuration**.
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.
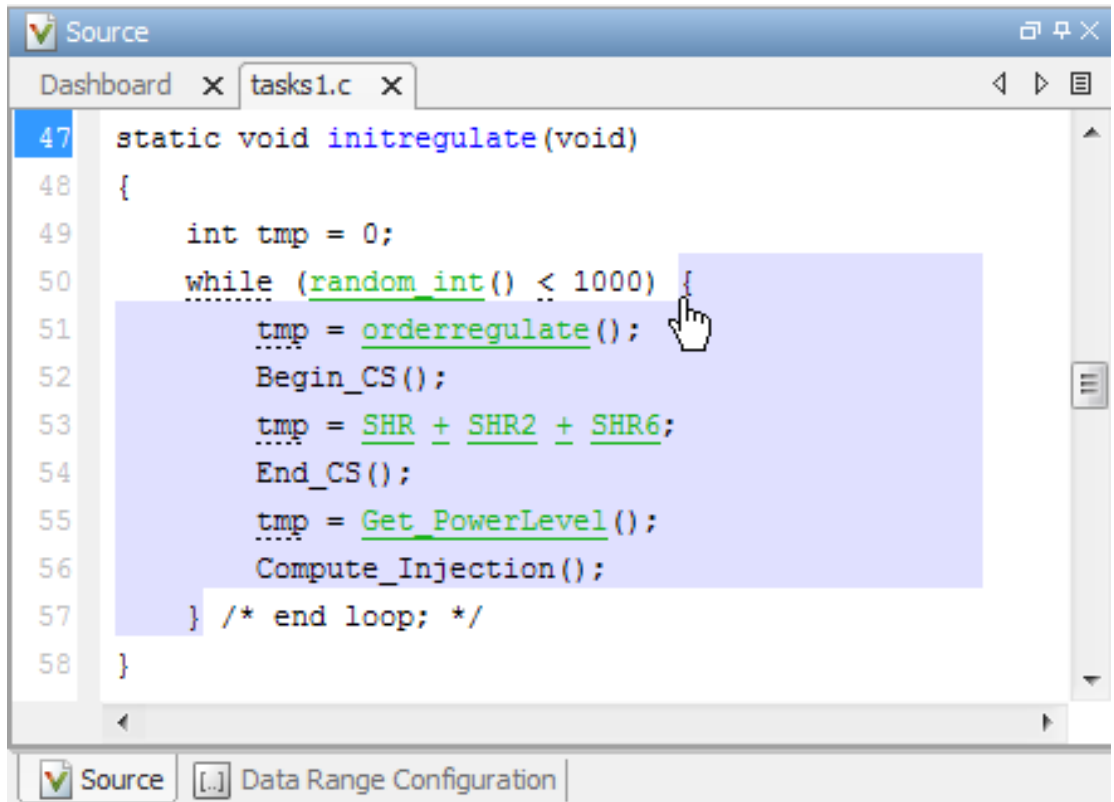
> **Note:** This action does not highlight the code block if the brace itself is already highlighted. The opening brace can be highlighted, for example, with a **Dead code** defect for the code block.

- 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. You can see:

- Code covered by analysis.
- Defect distribution. You can choose to view the distribution by:
  - **File**
  - **Category** or defect name.
- Distribution of coding rule violations. You can choose to view the distribution by:
  - **File**
  - **Category** or rule number.

  The **Dashboard** displays violations of different types of rules such as MISRA C, JSF® C++, or custom rules on different graphs.

  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 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 other instances of the variable.

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.

Function replacement in Simulink plug-in

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

Function	What Happens?	Use This Function Instead
`PolyspaceAnnotation`	Warning	`pslinkfun('annotations',...)`
`PolySpaceGetTemplateCFGFile`	Warning	`pslinkfun('gettemplate')`
`PolySpaceHelp`	Warning	`pslinkfun('help')`
`PolySpaceEnableCOMServer`	Warning	`pslinkfun('enablebacktomodel')`
`PolySpaceSpooler`	Warning	`pslinkfun('queuemanager')`
`PolySpaceViewer`	Warning	`pslinkfun('openresults',...)`
`PolySpaceSetTemplateCFGFile`	Warning	`pslinkfun('settemplate',...)`
`PolySpaceConfigure`	Warning	`pslinkfun('advancedoptions')`
`PolySpaceKillAnalysis`	Warning	`pslinkfun('stop')`
`PolySpaceMetrics`	Warning	`pslinkfun('metrics')`

For more information, see pslinkfun

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` does not check the configuration before an analysis.

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

For more information about **Check configuration**, see Check Simulink Model Settings.

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

In R2014a, 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, see Identify Errors in Simulink Models.

Additional analysis checkers

Polyspace Bug Finder can now check for two additional defects in C and C++:

- **Wrong allocated object size for cast** checks for memory allocations that are not multiples of the pointer size.
- **Line with more than one statement** checks for lines that have additional statements after a semicolon.

For more information, see Wrong allocated object size for cast and Line with more than one statement.

Data range specification support

Data range specification (DRS) is available with Polyspace Bug Finder. You can add range information to global variables.

You can also use DRS information with Polyspace Code Prover™. Similarly, you can use DRS information from Code Prover in Bug Finder.

For more information, see Inputs & Stubbing.

Polyspace binaries being removed

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

- `polyspace-report-generator.exe`
- `polyspace-results-repository.exe`
- `polyspace-spooler.exe`
- `polyspace-ver.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 analysis represents `float arr = 0.1;` as,

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

R2013b

Version: 1.0

New Features

Introduction of Polyspace Bug Finder

Polyspace Bug Finder is a new companion product to Polyspace Code Prover. Polyspace Bug Finder analyzes C and C++ code to find possible defects and coding rule violations. Bug Finder can run fast analyses on large code bases with low false-positive results. Polyspace Bug Finder also calculates code complexity metrics with Polyspace Metrics.

Bug Finder integrates with Simulink, Eclipse, Visual Studio, and Rhapsody® to help you analyze code from within your development environment.

Detection of run-time errors, data flow problems, and other defects in C and C++ code

Polyspace Bug Finder uses static analysis to find various defects for C and C++ code with few false-positive results. The analysis does not require program execution, code instrumentation, or test cases.

Some categories of defects are:

- Numeric
- Programming
- Static memory
- Dynamic memory
- Data-flow

To see a list of defects you can find, see Polyspace Bug Finder Defects.

Bug Finder analysis runs quickly, so you can fix errors and rerun analysis.

For information about running analyses, see Find Bugs.

Fast analysis of large code bases

Polyspace Bug Finder uses an efficient analysis method which produces results quickly, even from large code bases. Therefore you can fix errors and rerun the analysis without having to wait. You can find more issues early on in the development process and produce better quality code overall.

Compliance checking for MISRA-C:2004, MISRA-C++:2008, JSF++, and custom naming conventions

Polyspace Bug Finder can also check for compliance with coding rules. There are four industry-defined rules you can select:

- MISRA C
- MISRA AC-AGC
- MISRA C++
- JSF C++

In addition, you can define rules to check for naming conventions.

You can run the coding rules checker separately, or at the same time as your analysis.

For more information, see Check Coding Rules.

Cyclomatic complexity and other code metrics

Using Polyspace Metrics, Polyspace Bug Finder calculates various code metrics, including cyclomatic complexity. These statistics are displayed using Polyspace Metrics, an integrated Web interface. You can use these results to track code quality over time. You can also share the code metrics, allowing others to track your project's progress.

Eclipse integration

Polyspace Bug Finder comes with an Eclipse plug-in that integrates Polyspace into your development environment. You can set up options, run analyses, view results, and fix bugs in the Eclipse interface. Using the Polyspace plug-in, you can quickly find and fix bugs as you code.

For a tutorial on using the Polyspace Bug Finder plug-in, see Find Defects from the Eclipse Plug-In.

Traceability of code verification results to Simulink models

For generated code from Simulink models, Polyspace analysis results link directly back to your Simulink model. You can trace defects back to the block that is causing the bug.

In the Source Code view of the Results Manager, the block names appear as links. When you select a link, the corresponding block is highlighted in Simulink.

For a tutorial on using Polyspace Bug Finder with Simulink models, see Find Defects from Simulink.

Access to Polyspace Code Prover results

A Polyspace Bug Finder installation also includes the Polyspace Code Prover user interface. With only a Polyspace Bug Finder license, you cannot run local Polyspace Code Prover verifications in the Polyspace Code Prover interface. However, you can use the Polyspace Code Prover interface to review results and upload comments to Polyspace Metrics.

For more information, see the Polyspace Code Prover Documentation.