

Polyspace[®] Bug Finder[™]

Release Notes

How to Contact MathWorks



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



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

Polyspace® Bug Finder™ Release Notes

© COPYRIGHT 2013–2014 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.

R2014a

Automatic project setup from build systems	2
Classification of bugs according to the Common Weakness Enumeration (CWE) standard	2
Additional coding rules support (MISRA-C:2004 Rule 18.2, MISRA-C++ Rule 5-0-11)	3
Support for GNU 4.7 and Microsoft Visual Studio C++ 2012 dialects	3
Simplification of coding rules checking	3
Preferences file moved	5
Security level support for batch analysis	6
Interactive mode for remote analysis	6
Default text editor	6
Results folder appearance in Project Browser	7
Results manager improvements	8
Support for Windows 8 and Windows Server 2012	10
Function replacement in Simulink plug-in	11
Check model configuration automatically before analysis	11
Additional back-to-model support for Simulink plug-in ...	12
Additional analysis checkers	12
Data range specification support	12
Polyspace binaries being removed	13
Improvement of floating point precision	13

R2013b

Introduction of Polyspace Bug Finder	16
Detection of run-time errors, data flow problems, and other defects in C and C++ code	16
Fast analysis of large code bases	17
Compliance checking for MISRA-C:2004, MISRA-C++:2008, JSF++, and custom naming conventions	17
Cyclomatic complexity and other code metrics	17
Eclipse integration	18

Traceability of code verification results to Simulink	
models	18
Access to Polyspace Code Prover results	18

R2014a

Version: 1.1

New Features: Yes


Bug Fixes: Yes

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 . 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 has 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

Compatibility Considerations: Yes

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.	<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>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. 	<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.
Format of the custom coding rules file.	<p>Each line in the file must have the syntax:</p> <pre>rule off error warning #comments</pre>	<p>Each line in the file must have the syntax:</p> <pre>rule off warning #comments</pre>

Coding Rules Feature	R2013b	R2014a
	For example: <pre># MISRA configuration - Proj1 10.5 off #don't check 10.5 17.2 error 17.3 warning</pre>	For example: <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.

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:

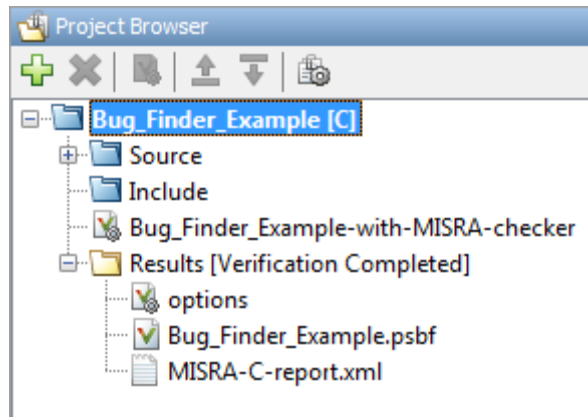
- WordPad in Windows
- vi in Linux

You can change the text editor on the **Editors** tab under **Options > Preferences**. For more information, see “Specify Text Editor”.

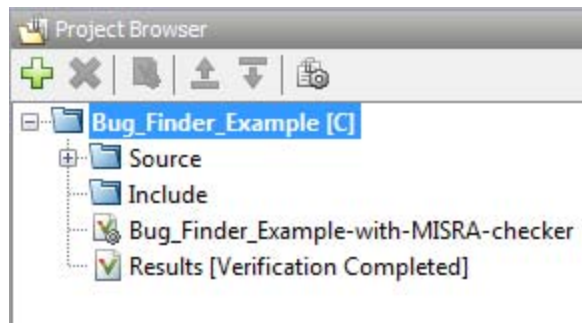
Results folder appearance in Project Browser

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 <code>.psbf</code> .	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.

```

Source
Dashboard code_unreachable.c
20
21 int main (void)
22 {
23     int i, res_end;
24     enumState inter;
25
26     res_end = State(Init);
27     if (res_end == 0) {
28         res_end = State(End);
29         inter = (enumState)intermediate_state(0);
30         if (res_end || inter == Wait) { // UNR code on inter== Wait
31             inter = End;
32         }
33         // use of I not initialized
34         if (random_int()) {
35             inter = (enumState)intermediate_state(i); // NIV ERROR
36             if (inter == Intermediate) { // UNR code because of NIV ERROR
37                 inter = End;
38             }
39         }
40     } else {
41         i = 1; // UNR code
42         inter = (enumState)intermediate_state(i); // UNR code
43     }
44     if (res_end) { // UNR code always reached, but no else
45         inter = End;
46     }
47
48     return res_end;
49 }
50

```

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

Compatibility Considerations: Yes

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
<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>
<code>PolySpaceSpooler</code>	Warning	<code>pslinkfun('queuemanager')</code>
<code>PolySpaceViewer</code>	Warning	<code>pslinkfun('openresults',...)</code>
<code>PolySpaceSetTemplateCFGFile</code>	Warning	<code>pslinkfun('settemplate',...)</code>
<code>PolySpaceConfigure</code>	Warning	<code>pslinkfun('advancedoptions')</code>
<code>PolySpaceKillAnalysis</code>	Warning	<code>pslinkfun('stop')</code>
<code>PolySpaceMetrics</code>	Warning	<code>pslinkfun('metrics')</code>

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

Compatibility Considerations: Yes

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: Yes

Bug Fixes: No

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](#).