Polyspace® Code Prover™ Release Notes

Contents

R2014a

Version: 9.1

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:

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

  Click 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".

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

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

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

For more information about these and other supported dialects, see Dialects for C or Dialects for C++.

Documentation in Japanese

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

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

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

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

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

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

Preferences file moved

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

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

For more information, see “Storage of Polyspace Preferences”.

Support for batch analysis security levels

When creating an MDCS server for Polyspace batch analyses, you can now add additional security levels through the **MATLAB Admin Center**. Using the **Metrics and Remote Server Settings**, the MDCS server is automatically set to security level zero. If you want additional security for your server, use the **Admin Center** button. The additional security levels require authentication by user name, cluster user name and password, or network user name and password.

For more information, see MDCS documentation.

Interactive mode for remote verification

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

To run verification in this mode

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

For more information, see “Interactive”.

Default text editor

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

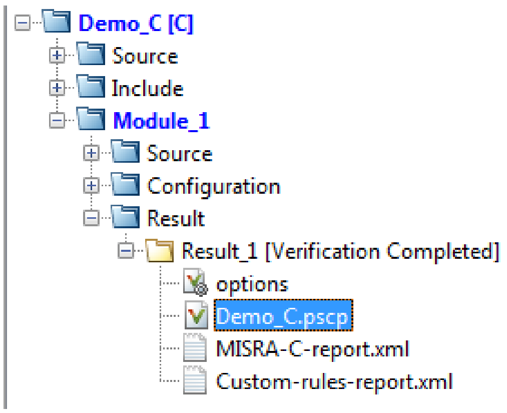
- WordPad in Windows
- vi in Linux

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

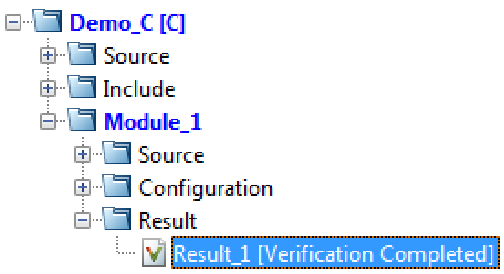
Results folder appearance in Project Browser

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

- Format before R2014a:



- Format in R2014a:

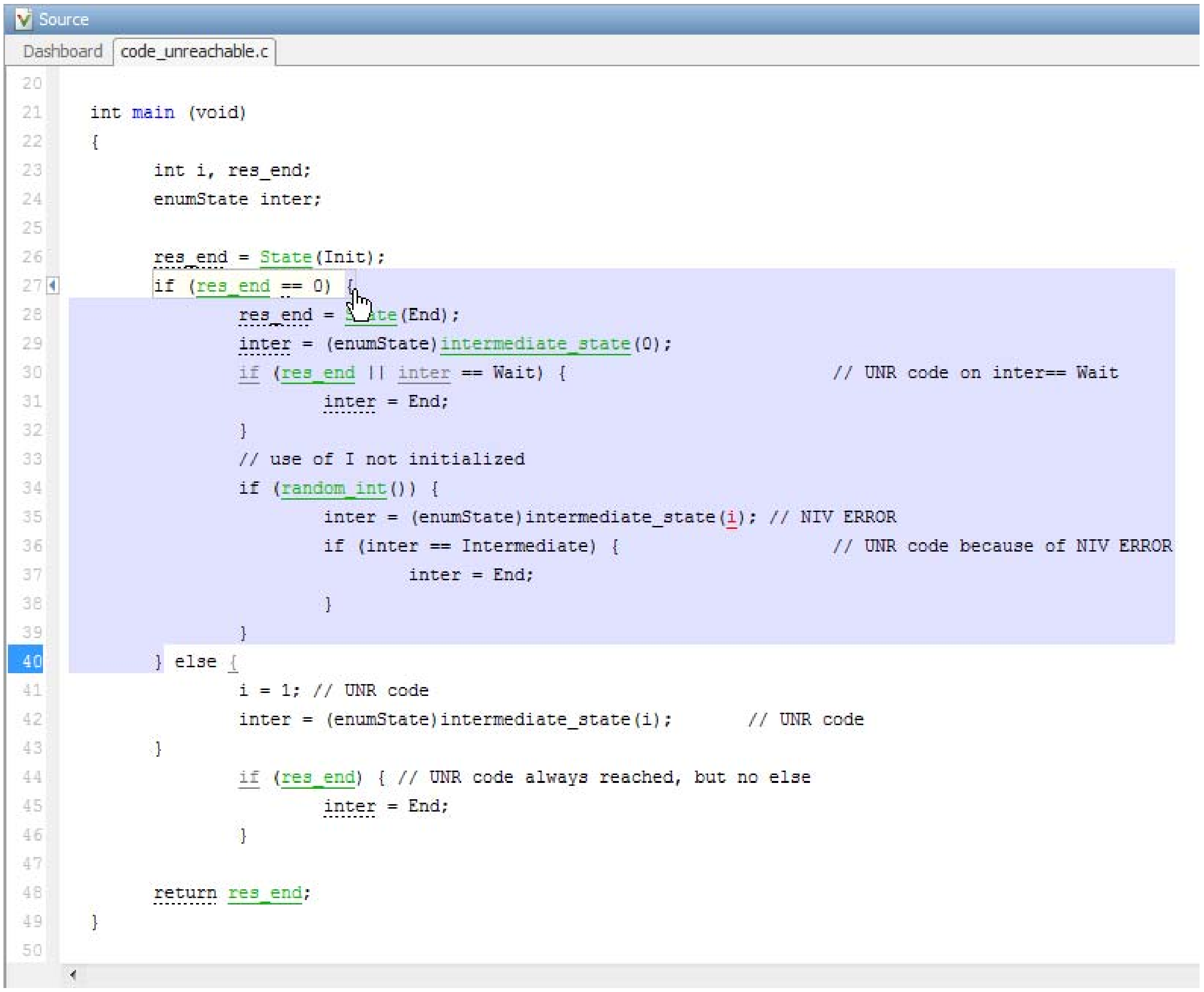


The following table lists the changes in the actions that you can perform on the results folder.

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



```

int main (void)
{
        int i, res_end;
        enumState inter;

        res_end = State(Init);
        if (res_end == 0) {
                res_end = State(End);
                inter = (enumState)intermediate_state(0);
                if (res_end || inter == Wait) {                       // UNR code on inter== Wait
                        inter = End;
                }
                // use of I not initialized
                if (random_int()) {
                        inter = (enumState)intermediate_state(i); // NIV ERROR
                        if (inter == Intermediate) {                  // UNR code because of NIV ERROR
                                inter = End;
                        }
                }
        } else {
                i = 1; // UNR code
                inter = (enumState)intermediate_state(i);       // UNR code
        }
                if (res_end) { // UNR code always reached, but no else
                        inter = End;
                }

        return res_end;
}

```

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

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

  On the **Dashboard**, you can obtain an overview of the results in a graphical format. For more information, see “Dashboard”.

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

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

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

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

Simplification of coding rules checking

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	2013b	2014a
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 verification will not stop even if the rule is violated. The rule violation will however be reported on the **Results Summary** pane.

- If you run analysis from the command line, the verification will stop if the rule is violated.

Support for Windows 8 and Windows Server 2012

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

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

Check model configuration automatically before analysis

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

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

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

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

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

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

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

Function replacement in Simulink plug-in

Compatibility Considerations: Yes

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

Function	What Happens?	Use This Function Instead
`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')`

Polyspace binaries being removed

Compatibility Considerations: Yes

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

- `polyspace-automatic-orange-tester.exe`
- `polyspace-c.exe`
- `polyspace-cpp.exe`

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

Improvement of floating point precision

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

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

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

R2013b

Version: 9.0

New Features: Yes

Bug Fixes: No

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

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

Color-coding of run-time errors directly in code

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

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

Errors detected include:

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

For more information, see Interpret Results.

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

Polyspace Code Prover calculates and displays range information associated with, for example, variables, function parameters and return values, and operators. The displayed range information represents a superset of dynamic values, which the software computes using static methods.

For more information, see Interpret Results.

Identification of variables exceeding specified range limits

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

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

You can use DRS to set constraints on:

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

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

For more information, see Data Range Configuration.

Quality metrics for tracking conformance to software quality objectives

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

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

For more information, see Quality Metrics.

Web-based dashboard providing code metrics and quality status

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

For more information, see Quality Metrics.

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

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

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

  For example, if you are reviewing verification results for the first time, select **First checks to review**. The software displays all red and gray checks but only a subset of orange checks. These orange checks are the ones most likely to be run-time errors. For more information, see Review Checks Using Predefined Methodologies.
- You can group checks by `File/Function` or `Check`:
  - Grouping by `Check` classifies checks by color. Within each color, this grouping classifies checks by categories related to the origin of the check, such as `Control flow`, `Data flow`, and `Numerical`.
  - Grouping by `File/Function` classifies checks by the file where they originated. Within each file, this grouping classifies checks by functions where they originated.
  - For C++ files, you can also group checks by `Class`. This grouping classifies checks by the class definition where they originated.

  For more information, see Organize Check Review Using Filters and Groups.
- You can filter checks using any of the column information criteria on the **Results Summary** pane. For example, you can filter out checks that you have already justified using the filter icon on the **Justified** column header. If you have applied a filter, the column heading changes to indicate that all results are not displayed. You can also define custom filters. For more information, see Organize Check Review Using Filters and Groups.
- You can navigate through the **Results Summary** pane using the keyboard or UI buttons. Both means of navigation respect the grouping, filters, and methodology used to display results.

Graphical display of variable reads and writes

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

For more information, see Exploring Results Manager Perspective.

Comparison with R2013a Polyspace products

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

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

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

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

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

Remote verification

Category	R2013a	R2013b
Products required	Install: • Polyspace Client for C/C++ on local computer • Polyspace Server for C/C++ on network computers, which are configured as Queue Manager and CPUs.	Install: • MATLAB, Polyspace Bug Finder, and Parallel Computing Toolbox on local computer. • MATLAB, Polyspace Bug Finder, Polyspace Code Prover, and MATLAB Distributed Computing Server on head node of computer cluster. For information about setting up a cluster, see Install Products and Choose Cluster Configuration.

Category	R2013a	R2013b
Configuring and starting services	On the **Polyspace Preferences > Server Configuration** tab: • Under **Remote configuration**, specify host computer for Queue Manager and Polyspace Metrics server and communication port. • Under **Metrics configuration**, specify other settings for Polyspace Metrics.	On the **Polyspace Preferences > Server Configuration** tab: • Under **MDCS cluster configuration**, specify computer for cluster head node, which hosts the MATLAB job scheduler (MJS). The MJS replaces the R2013a Polyspace Queue Manager. • Under **Metrics configuration**: - Specify host computer for Polyspace Metrics server and communication port. - Specify other settings for Polyspace Metrics.
	In the Remote Launcher Manager dialog box: 1 Under **Common Settings**, specify Polyspace communication port, user details, and results folder for remote verifications. 2 Under **Queue Manager Settings**, specify Queue Manager and CPUs. 3 Under **Polyspace Server Settings**, specify available Polyspace products.	In the Metrics and Remote Server Settings dialog box: 1 Under **Polyspace Metrics Settings**, specify user details, Polyspace communication port, and results folder for remote verifications. 2 Under **Polyspace MDCS Cluster Security Settings**, you see the following options with default values: • **Start the Polyspace MDCE service** — Selected. The `mdce`

Category	R2013a	R2013b
	4 To start the Queue Manager and Polyspace Metrics service, click **Start Daemon**.	service, which is required to manage the MJS, runs on the MJS host computer and other nodes of the cluster. • **MDCE service port** — 27350. • **Use secure communication** – Not selected. Communication is not encrypted. You may want to use communication with security. For information about MATLAB Distributed Computing Server cluster security, see Cluster Security. **3** To start the Polyspace Metrics and `mdce` services, click **Start Daemon**. Use the Metrics and Remote Server Settings dialog box to start and stop `mdce` services only if you configure the MDCS head node as the Polyspace Metrics server. Otherwise, clear the **Start the Polyspace MDCE service** check box, and use the MDCS Admin Center. To open the MDCS Admin Center, run: `MATLAB_Install/toolbox/distcomp/bin/admincenter` For information about the MDCS Admin Center, see Cluster Processes and Profiles.

Category	R2013a	R2013b
Running a remote verification	In the Project Manager perspective: 1 On the **Configuration > Machine Configuration** pane, select the following check boxes: • **Send to Polyspace Server** • **Add to results repository** — Allows viewing of results through Polyspace Metrics. 2 On the toolbar, click **Run**. The Polyspace client performs code compilation and coding rule checking on the local, host computer. Then the Polyspace client submits the verification to the Queue Manager on your network.	In the Project Manager perspective: 1 On the **Configuration > Distributed Computing** pane, select the **Batch** check box. By default, the software selects the **Add to results repository**, which enables the generation of Polyspace Metrics. 2 On the toolbar, click **Run**. The Polyspace Code Prover software performs code compilation and coding rule checking on the local, host computer. Then the Parallel Computing Toolbox client submits the verification job to the MJS of the MATLAB Distributed Computing Server cluster.
Managing remote verifications	Use the Queue Manager to monitor and manage submitted jobs from Polyspace clients. On the Web, you can monitor jobs through Polyspace Metrics. If you have installed Polyspace Server for C/C++ on your local computer, through Polyspace Metrics, you can open the Queue Manager .	Use the Queue Manager to monitor and manage jobs submitted through Parallel Computing Toolbox clients.
Accessing results of remote verifications	When you run a verification on a Polyspace server, the Polyspace software automatically downloads the results to your local, client computer. You can view the results in the Results Manager perspective.	On the Web, use Polyspace Metrics to view verification results. If Polyspace Bug Finder is installed on your local computer, you can download verification results. For example, in Polyspace Metrics,

Category	R2013a	R2013b
	In addition, you can use the Queue Manager to download results of verifications submitted from other Polyspace clients.On the Web, use Polyspace Metrics to view verification results stored in results repository. If Polyspace Client for C/C++ is installed on your local computer, you can download verification results. For example, in Polyspace Metrics, clicking a cell value in the **Run-Time Checks** view opens the corresponding verification results in the Results Manager.	clicking a **Project** cell in the **Runs** view opens the corresponding verification results in the Results Manager.

Results review

Category	R2013a	R2013b
Results Explorer	Available. Allows navigation through checks by the file and function where they occur. To view, select **Window > Show/Hide View > Results Explorer**.	Removed. To navigate through checks by file and function, on **Results Summary** pane, from the drop-down menu, select `File/Function`.
Filters on the **Results Summary** pane	Filters appear as icons on the **Results Summary** pane. You can filter by: • Run-time error category • Coding rules violated • Check color • Check justification • Check classification	You can filter by the information in all the columns of the **Results Summary** pane. In addition to existing filters, the new filtering capabilities extend to the file, function and line number where the checks appear. You can also define your own filters.

Category	R2013a	R2013b
	• Check status	The filters appear as the icon on each column header. To apply a filter using the information in a column: **1** Place your cursor on the column header. The filter icon appears. **2** Click the filter icon and from the context menu, clear the **All** box. Select the appropriate boxes to see the corresponding checks. For more information, see Organize Check Review Using Filters and Groups.
Code Coverage Metrics	In the **Results Explorer** view, the software displays two metrics for the project: • `unp` — Number of unreachable functions as a ratio of total number of functions • `cov` — Percentage of elementary operations covered by verification The unreachable procedures are marked gray in the **Results Explorer** view.	The new **Results Statistics** pane displays the code coverage metrics through the **Code covered by verification** column graph. To see a list of unreachable procedures, click this column graph. For more information, see Results Statistics.

Other features

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

Product	Feature	R2013a	R2013b
	Run verifications from multiple Polyspace environments	Supported	Not supported, produces a license error `-4,0`.
	Non-official options field	Available in **Configuration > Machine Configuration** pane	Renamed **Other** and moved to **Configuration > Advanced Settings** pane
Polyspace Model Link SL and TL	Default includes	Includes specific to the target specified.	Generic includes for C and C++. These includes are target independent.
	Running a verification	**Code > Polyspace > Polyspace for Embedded Coder/Target Link** • Verify Generated Code • Verify Generated Model Reference Code Also right-clicking on a subsystem and selecting **Polyspace > Polyspace for Embedded Coder/Target Link**	**Code > Polyspace > Verify Code Generated for** • Selected Subsystem • Model • Referenced Model • Selected Target Link Subsystem Also right-clicking on a subsystem and selecting **Polyspace > Verify Code Generated for > Selected Subsystem / Selected Target Link Subsystem**
	Product Mode	Not available.	Choose between `Code Prover` or `Bug Finder` depending on the type of analysis you want to run.

Product	Feature	R2013a	R2013b
	Settings	Available. Called **Verification Settings from**	Available. Called **Settings from**. Functionality the same.
	Open results	Option **Open Project Manager and Results Manager** opened the Polyspace Project Manager.	Option **Open results automatically after verification** opens Polyspace Metrics (batch verifications) or Polyspace Results Manager (local verifications).
Polyspace plug-in for Visual Studio 2010	Support for C++11 features	Partial support.	Added support for: • Lambda functions • Rvalue references for `*this` and initialization of class objects by rvalues • Decltype • Auto keyword for multi-declarator auto and trailing return types • Static assert • Nullptr • Extended friend declarations • Local and unnamed types as template arguments

Options

Product	Option	R2013a	R2013b
	`-code-metrics`	Available. Not selected by default.	Removed. Code complexity metrics computed by default.
	`-dialect`	Available.	Default unchanged, but new value `gnu4.6` available for C and C++.
Polyspace Client and Server for C/C++	`-max-processes`	Specify through **Machine Configuration > Number of processes for multiple CPU core systems** or command line .	Specify from command line, or through **Advanced Settings > Other**.
	`-allow-language-extensions`	Available. Selected by default.	Removed. By default, software supports subset of common C language constructs and extended keywords defined by the C99 standard or supported by many compilers.
	`-enum-type-definition`	Available with three values. First value called `defined-by-standard`.	Available with three values. For C, first value renamed `signed-int`. For C++, first value renamed `auto-signed-int-first`.

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