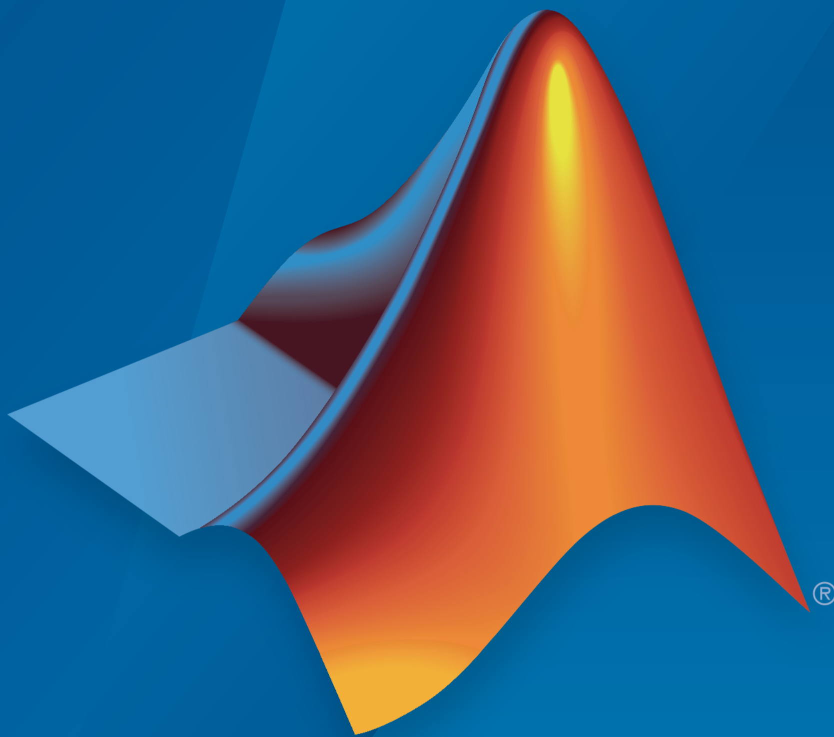
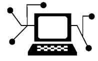
Fixed-Point Designer™ Release Notes



MATLAB®

MathWorks®

How to Contact MathWorks



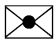
Latest news: www.mathworks.com

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

Fixed-Point Designer™ Release Notes

© COPYRIGHT 2013–2017 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.

Contents

R2017a

Simulink Diagnostic Management: Control which simulation and fixed-point diagnostic warnings you receive from specific blocks, including model reference **1-2**
Select blocks with certain diagnostic suppressions by default . **1-2**
Diagnostic suppressor functions support `MSLDiagnostic` as input argument **1-2**
Improved workflow for suppressing diagnostics from referenced models **1-3**

Derived range analysis support for System objects in Simulink **1-3**

Autoscaling support for Simulink.AliasType objects **1-3**

Improved data type proposals for shared data type groups across model reference **1-3**

More fixed-size variable information in Convert to Fixed-Point step of the Fixed-Point Converter app **1-4**

`fimath` property changes **1-5**

R2016b

Single-Precision Conversion: Automatically convert double-precision systems to use single-precision data types in Simulink **2-2**

Float to Fixed Conversion of MATLAB Function Blocks: Automatically generate fixed-point versions of floating-point MATLAB Function blocks 2-2

Histogram Instrumentation in Simulink: Generate log2 histograms of Simulink signals and blocks from simulation data 2-4

Autoscaling numerictype Objects: Propose and apply fixed-point data types for Simulink numeric type objects 2-7

Range analysis support for FIR filters, Dead Zone, and Rate Limiter blocks 2-7

Simulink Diagnostic Suppressor 2-7

Reduced number of multiplication helper functions 2-8

Improved accuracy of fixed-point sin, cos, and mod functions 2-8

Improved workflow for collecting and analyzing ranges in the Fixed-Point Converter app 2-9

R2016a

Autoscaling Parameter Objects: Automatically propose and apply data types for parameter objects 3-2

View and edit `fi` objects in Model Explorer 3-2

Simulate system level designs that integrate referenced models targeting an assembly of heterogeneous embedded devices 3-3

Enhancements to Fixed-Point Converter app 3-4
- Support for arrays of structures 3-4
- Structures in generated fixed-point code 3-4
- Revert changes to input type definitions 3-4

View complete error message in error table 3-4
Additional keyboard shortcuts in the code generation report . 3-4
Changes to Fixed-Point Conversion Code Coverage 3-5

R2015aSP1

Bug Fixes

R2015b

Simulink Fixed-Point Tool workflow simplification: Propose signedness and data types for inherited and floating-point types **5-2**
System under design (SUD) specification 5-2
Signedness proposals 5-2
Proposals for objects using inherited and floating-point types 5-3
Two-way traceability between model and Fixed-Point Tool . . 5-3
New configurations for model settings 5-4

Double-precision to single-precision conversion: Convert double-precision MATLAB code to single-precision MATLAB code using the command line **5-4**

MATLAB Fixed-Point Converter app streamlined workflow: Restore project state and minimize regeneration of MEX files **5-5**
Saving and restoring fixed-point conversion workflow state in the app 5-5
Minimized regeneration of MEX files 5-5
Specification of additional `fimath` properties in app editor . . 5-6
Improved management of comparison plots 5-6
Variable specializations 5-7
Improvements to Readability of Generated Code 5-8
Tab completion for specifying files 5-9
Improvements for manual type definition 5-9

Compatibility between the app colors and MATLAB preferences 5-10

Range analysis for Delay blocks: Improve accuracy and speed of range analysis on models using Delay blocks **5-10**

Control of signed shifts in fixed-point scaling operations: Control the use of signed shifts in generated code **5-11**
MATLAB **5-11**
Simulink **5-12**

Access full-precision value of `fi` object in decimal and string format **5-12**

Detection of multiword operations **5-12**
MATLAB **5-12**
Simulink **5-13**

Enhanced Model Advisor check for implementing strict single-precision designs **5-13**

System object instrumentation in Fixed-Point Tool **5-13**

R2015a

Derived Ranges for MATLAB Function Blocks in Simulink . **6-2**

Fixed-Point Converter app enhancements, including detection of dead and constant folded code, support for projects with multiple entry point functions and support for global variables **6-2**
Support for projects with multiple entry-point functions **6-2**
Support for global variables **6-2**
Code coverage based translation **6-2**
Conversion from project to MATLAB scripts for command-line fixed-point conversion **6-3**
Generated fixed-point code enhancements **6-3**
Integration with MATLAB Coder app interface **6-3**

Automated conversion of additional DSP System objects using the Fixed-Point Converter app 6-3

Fixed-Point SimState logging and root logging improvements 6-4

Flexible structure assignment of buses 6-4

`eye(m,'like',a)` syntax supported for fixed-point inputs . . 6-4

New interpolation method for generating lookup table MATLAB function replacements 6-4

Fixed-point scaling information in Code Interface Report . . 6-5

R2014b

Fixed-Point Converter app for automated conversion of floating-point MATLAB code 7-2

Commands for scripting fixed-point conversion and accessing the collected data in Simulink 7-2

Automated fixed-point conversion for commonly used DSP System objects, including Biquad Filter, FIR Filter, and FIR Rate Converter 7-3

Simulation range collection and data type proposals for MATLAB Function blocks in Simulink 7-3

Overflow diagnostics to distinguish between wrap and saturation in Simulink 7-3

Highlighting of potential data type issues in generated HTML report 7-4

Code generation of for loops using fixed-point loop indices . 7-4

Cast net slope computations using rational numbers 7-4

Lock Column View option in the Fixed-Point Tool **7-4**

Fixed-Point Advisor enhancements **7-5**

hdlram renamed hdl.RAM **7-5**

Changes to data type strings **7-5**
Signal data type display **7-5**
tostring function now uses 0 and 1 to represent signedness . . **7-5**

New featured examples **7-6**

R2014a

Data type override and automatic data typing for bus objects **8-2**
Data type override for bus objects **8-2**
Autoscaling for bus objects **8-2**

Derived ranges for complex signals in Simulink **8-2**

cordicsqrt function for fixed-point CORDIC-based square root functionality **8-2**

Overflow detection with scaled double data types in MATLAB Coder projects **8-3**

Fixed-point ARM Cortex-M code replacement support for DSP System Toolbox FIR filters **8-3**

Fixed-Point Advisor support for referenced configuration sets **8-3**

Enhancements to automated conversion of MATLAB code . **8-3**
Support for MATLAB classes **8-3**
Generated fixed-point code enhancements **8-4**
Fixed-point report **8-4**

Automatic C compiler setup **8-4**

More flexible control of dsp.LMSFilter System object fixed-point settings **8-4**

Derived ranges for For Each and For Each Subsystem blocks **8-5**

R2013b

C99 long long integer data type for embedded code generation **9-2**

Model Advisor fixed-point checks with additional coverage and optimization awareness **9-2**

`fi` object as an index in colon expressions and an argument to numel and bit index functions **9-3**
fi object as an index in colon expressions **9-3**
fi objects as bit index input argument **9-3**
fi objects as shift-value input argument **9-3**
numel function support for fi inputs **9-3**

Improved efficiency of data type internal rules for Lookup Table blocks **9-3**

Derived ranges for complex variables in MATLAB Coder projects **9-4**

Simplified modeling of single-precision designs **9-4**

Range analysis support on Mac platforms **9-5**

Changes to showInstrumentationResults function options . **9-5**
New option to suppress display of MATLAB code **9-5**
Removal of `-browser` option **9-5**

Changes to Continuous state-space block family range analysis support **9-5**

Enhanced fiaccel support for int64 and uint64 functions . . . **9-6**

Support for LCC compiler on Microsoft Windows (64-bit) machines 9-6

Warning for use of inexact `fi` and `fimath` property names . . 9-6

Conversion of numeric variables into Simulink.Parameter objects 9-6

Fixed-point conversion test file coverage results 9-7

Fixed-point conversion workflow supports designs that use enumerated types 9-7

Fixed-point conversion of variable-size data using simulation ranges 9-7

Error checking improvements for bitconcat, bitandreduce, bitorreduce, bitxorreduce, bitsliceget functions 9-7

Legacy data type specification functions return numeric objects 9-8

numberofelements function being removed in a future release 9-10

R2013a

Product restructuring 10-2

Histogram logging in instrumented MATLAB Code Generation report 10-2

`fi` object in indexing and switch-case expressions 10-2

zeros, ones, and cast code reuse for floating-point and fixed-point types 10-2

Code generation for `x.^n` when n is a variable and `x` is a `fi` object 10-4

Fixed-Point Advisor support for model reference **10-4**

Automated conversion of floating-point to fixed-point types in MATLAB Coder projects **10-4**

Improved autoscaling for models with virtual bus signals . **10-5**

Data Type Override for MATLAB Function block using built-in doubles and singles **10-5**

Instrumentation for arrays of structs **10-5**

File I/O function support **10-5**

Support for nonpersistent handle objects **10-6**

Load from MAT-files for code acceleration **10-6**

New toolbox functions supported for code acceleration and generation **10-6**

Function to be removed in a future release **10-7**

Function being removed **10-8**

R2017a

Version: 5.4

New Features

Bug Fixes

Compatibility Considerations

Simulink Diagnostic Management: Control which simulation and fixed-point diagnostic warnings you receive from specific blocks, including model reference

Select blocks with certain diagnostic suppressions by default

Beginning in R2017a, the Counter Free-Running, HDL Counter, Counter Limited, and Extract Bits blocks no longer report wrap on overflow warnings. The blocks continue to report errors due to wrap on overflows. You can restore the warning diagnostic by breaking the library link and using the `Simulink.restoreDiagnostic` function.

Diagnostic suppressor functions support `MSLDiagnostic` as input argument

You can now suppress and restore certain diagnostic warnings thrown by your model using a `Simulink.MSLDiagnostic` object as an input to the `Simulink.suppressDiagnostic` and `Simulink.restoreDiagnostic` functions.

To use simulation metadata and `MSLDiagnostic` objects, use `set_param` to set `ReturnWorkspaceOutputs` to `on`. Store the simulation output in a variable.

```
set_param(model_name,'ReturnWorkspaceOutputs','on');
out = sim(model_name);
```

Access the `MSLDiagnostic` object through the simulation output.

```
diag = out.getSimulationMetadata.ExecutionInfo.WarningDiagnostics(1).Diagnostic
```

```
diag =

  MSLDiagnostic with properties:

    identifier: 'SimulinkFixedPoint:util:fxpParameterPrecisionLoss'
       message: 'Parameter precision loss occurred for 'Value' of 'Suppressor_CLI_Demo/
         paths: {'Suppressor_CLI_Demo/one'}
         cause: {}
         stack: [0×1 struct]
```

Use the `Simulink.suppressDiagnostic` function to suppress the diagnostic warning specified by the `MSLDiagnostic` object, `diag`.

```
Simulink.suppressDiagnostic(diag)
```

You can restore the diagnostic using the `Simulink.restoreDiagnostic` function

```
Simulink.restoreDiagnostic(diag)
```

Improved workflow for suppressing diagnostics from referenced models

You can now suppress certain diagnostic warnings for specified instances of warnings in a referenced model. By accessing the `MSLDiagnostic` object of the specific instance of the warning, you can suppress the warning only when the block inside the referenced model is simulated from the specified top model.

Derived range analysis support for System objects in Simulink

Using the Fixed-Point Tool, you can now derive ranges for models that use handle objects, including System objects. For more information on range analysis in the Fixed-Point Tool, see "How Range Analysis Works".

Autoscaling support for `Simulink.AliasType` objects

Using the Fixed-Point Tool, you can now propose and apply data types for `Simulink.AliasType` objects used in your model. The Fixed-Point Tool detects alias type objects in your model and proposes a fixed-point data type based on their respective values and ranges. The tool applies the proposed data type to the alias type object by updating the definition of the object in the base workspace. For more information, see "Autoscale Simulink.AliasType Objects".

Improved data type proposals for shared data type groups across model reference

In past releases, there was limited traceability of model objects which were required to use the same data type across model reference boundaries. This often resulted in an update diagram error after applying proposed data types.

Beginning in R2017a, when the Fixed-Point Tool proposes data types for data objects in shared data type groups, the tool generates a proposal based on all collected ranges, including range information from data objects used inside referenced models. The Fixed-Point Tool can also now highlight all model elements that must use the same data type when the shared data type group crosses model reference boundaries.

More fixed-size variable information in Convert to Fixed-Point step of the Fixed-Point Converter app

In R2017a, in the Fixed—Point Converter app, after you convert floating-point MATLAB® code to fixed-point MATLAB code, the app provides fixed-point type information for variables.

Variables | Function Replacements | Output

Variable	Type	Size	Signed	Word Length	Fraction Length
⊟ Input					
x	embedded.fi	1 x 256	Yes	16	14
⊟ Output					
y	embedded.fi	1 x 256	Yes	16	14
⊟ Persistent					
z	embedded.fi	2 x 1	Yes	16	15
⊟ Local					

In the code pane of the **Convert to Fixed-Point** step, after fixed-point conversion, if you place your cursor over a converted variable or expression, the app displays the fixed-point type information.

```
y = fi(zeros(size(x)), 1, 16, 14,
for i=1:length(x)
    y(i
    z(1                                + z
    z(2                                  )
end
nd
```

TYPE | FIMATH

Type: 1 x 256 embedded.fi

Signedness:	Signed
Word Length:	16
Fraction Length:	14

For a variable with a fixed-point type in the original code, when you place your cursor over the variable before or after conversion, the app displays the fixed-point type information.

`fimath` property changes

All `fimath` property names are case-sensitive and require that you use the full property names. You cannot use truncated property names. In previous releases, when using truncated property names, a warning would appear. Beginning in R2017a, inexact property names result in an error.

Compatibility Considerations

To avoid seeing errors for `fimath` properties, update your code so it uses the full names and correct cases of all `fimath` properties. The full names and correct cases of the properties appear when you display a `fimath` object on the MATLAB command line.

R2016b

Version: 5.3

New Features

Bug Fixes

Single-Precision Conversion: Automatically convert double-precision systems to use single-precision data types in Simulink

Using the Single Precision Converter, you can now automatically convert Simulink® models from double-precision to single-precision. The Converter makes these changes:

- Conversion of user-specified double-precision data types to single-precision data types (applies to block settings, Stateflow chart settings, signal objects, and bus objects.)
- Output signals and intermediate settings using inherited data types which compile to double-precision change to single-precision data types.

The converter does not change Boolean, built-in integer, or user-specified fixed-point data types. When the conversion is finished, the converter displays a table summarizing the compiled and proposed data types of the objects in the system under design. When the conversion is finished, a table summarizes the compiled and proposed data types of the objects in the system under design.

To use the Single-Precision Converter, from the Simulink **Analysis** menu, select **Data Type Design** > **Single Precision Converter**. Under **System under design**, select the system to convert to single-precision, then click **Convert to Single**.

For more information, see Getting Started with Single Precision Converter.

Float to Fixed Conversion of MATLAB Function Blocks: Automatically generate fixed-point versions of floating-point MATLAB Function blocks

When converting a model that contains MATLAB Function blocks, you can now inspect type information of the MATLAB variables in the context of the code. This new code view provides a similar workflow to the Fixed-Point Converter app in MATLAB. To open the new code view, in the Fixed-Point Tool, under **Automatic Data Typing**, click **Inspect MATLAB Function blocks**.

Automatic data typing

Propose: Signedness ◯ Word length ◉ Fraction length

Propose for: Inherited Floating point

Default word length: 16

When proposing types use: All collected ranges

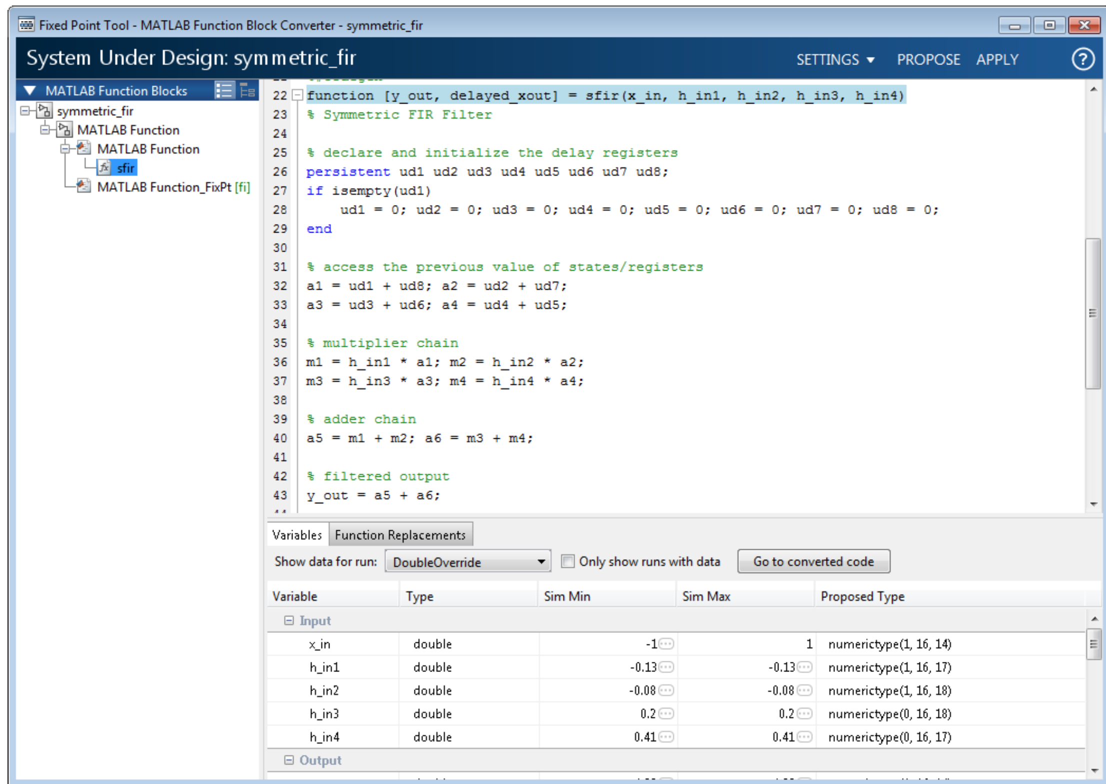
Safety margin for simulation min/max (%): 0

Inspect MATLAB Function blocks

DT Propose data types

DT Apply accepted data types

The window that opens helps you to inspect advanced conversion settings such as `fimath` settings, and MATLAB function replacements.



Once you are satisfied with the proposed data types, click **Apply** to have the tool automatically generate a variant subsystem. The variant subsystem contains the original floating-point version of the MATLAB function block, and a fixed-point version of the block. You can refine the conversion by modifying the original floating-point MATLAB code. The fixed-point variant will automatically update after reconverting the block.

Histogram Instrumentation in Simulink: Generate log2 histograms of Simulink signals and blocks from simulation data

Using the Fixed-Point Tool, you can now view a histogram of bits used by each object in your system under design. The bit weights are displayed along the X-axis, and the percentage of occurrences along the Y-axis. Each bin in the histogram corresponds to a bit in the binary word. The plot also includes the number of times that zero occurred. After simulating a system with fixed-point instrumentation or signal logging turned on,

select an object in your model from the Contents pane of the Fixed-Point Tool and select the **Result Details** tab to view the histogram plot.

Workflow | Result Details

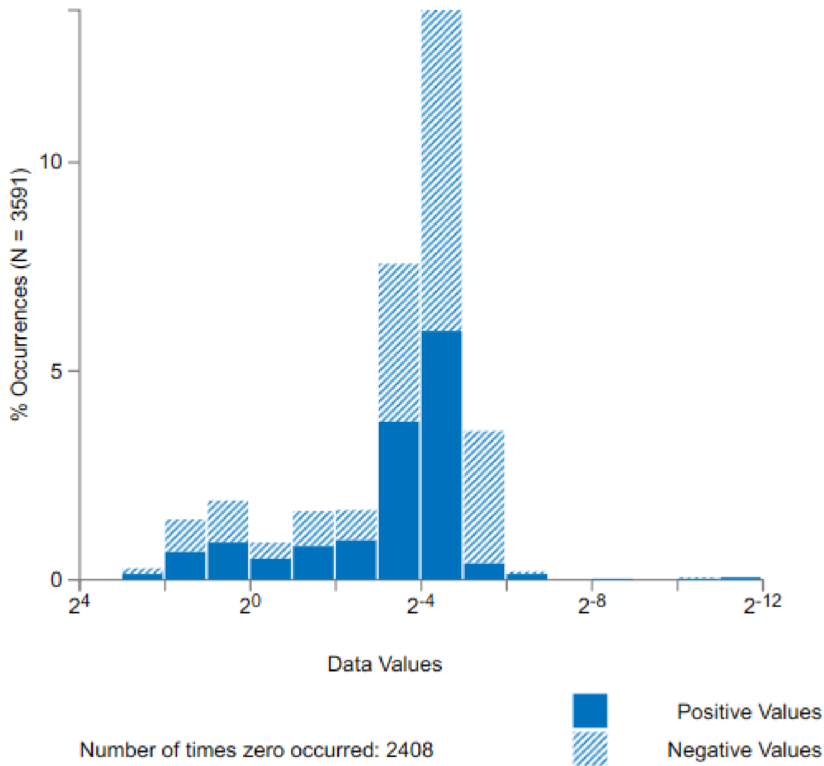
fxpdemo_feedback/Controller/Numerator Terms : Accumulator

Property	Data Type	Minimum	Maximum	Precision
SpecifiedDT	fixdt(1,32,28)	-8	7.99999999627471	3.7252902984619...

Range Information

Property	Minimum	Maximum
Simulation	-5.765953063964844	5.789508819580078

Histogram of Simulation Data



Autoscaling numerictype Objects: Propose and apply fixed-point data types for Simulink numeric type objects

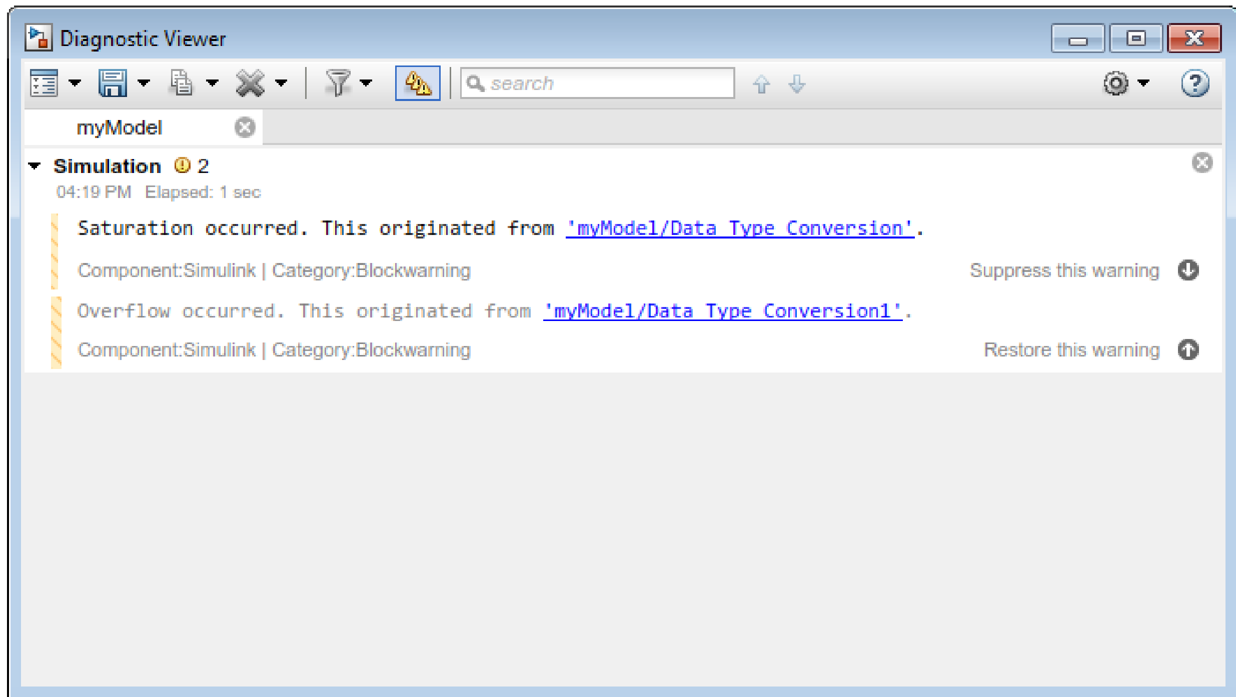
Using the Fixed-Point Tool, you can now propose and apply data types for `Simulink.NumericType` and `embedded.numerictype` objects used in your model. The Fixed-Point Tool detects numeric type objects in your model and proposes a fixed-point data type based on their respective values and ranges. The tool applies the proposed data type to the numeric type object by updating the definition of the object in the base or model workspace. For more information on autoscaling `Simulink.NumericType` objects, see Autoscale Simulink.NumericType Objects.

Range analysis support for FIR filters, Dead Zone, and Rate Limiter blocks

Using the Fixed-Point Tool, you can now derive ranges for models that use Discrete FIR Filter, Dead Zone, and Rate Limiter blocks. For more information on range analysis in the Fixed-Point Tool, see How Range Analysis Works.

Simulink Diagnostic Suppressor

The Diagnostic Viewer in Simulink now includes an option to suppress certain diagnostics. This feature enables you to suppress warnings for specific objects in your model. Click the **Suppress this warning** button next to the warning in the Diagnostic Viewer to suppress the warning from the specified source. You can restore the warning from the source by clicking **Restore this warning**.



You can also control the suppressions from the command line. For more information, see Suppress Diagnostic Messages Programmatically.

Reduced number of multiplication helper functions

When you generate code for your model, there are now fewer generated multiplication helper functions. The new multiplication helper functions parameterize the shift amount for multiplication operations using binary-point scaling, reducing the need for separate functions in the generated code.

This change results in reduced memory consumption. This reduction in the amount of code generated from a model aids in the maintainability of your code base.

Improved accuracy of fixed-point `sin`, `cos`, and `mod` functions

The fixed-point `sin` and `cos` functions are now more precise. In past releases these calculations were accurate only to within the top 16 most-significant bits of the input.

The `mod` function now has improved accuracy because it no longer limits internally-computed intermediate types to 32-bits or less.

For more information, see the sin, cos, and mod reference pages.

Improved workflow for collecting and analyzing ranges in the Fixed-Point Converter app

The **Simulate** and **Derive** buttons on the **Convert to Fixed Point** page of the Fixed-Point Converter app are now simplified and merged into a single **Analyze** button. This button controls which ranges (simulation ranges, design ranges, and derived ranges) are collected and used in the data type proposal phase of the conversion. When either the **Specify design ranges** or the **Analyze ranges using derived range analysis** options are selected, the **Static Min** and **Static Max** columns appear in the table. These columns do not appear when only the **Analyze ranges using simulation** option is selected, simplifying the view of the data. As in previous releases, you can still control which ranges are used for data type proposal in the **Settings** pane.

Fixed-Point Converter - ex_2ndOrder_filter.prj

Convert to Fixed Point | SETTINGS ▾ | ANALYZE ▾ | CONVERT | TEST ▾

Source Code

ex_2ndOrder_filter

☑ Analyze ranges using simulation ☑ Specify design ranges

Test bench ex_2ndOrder_filter_test.m ☐ Log data for histogram ☑ Show code coverage

☑ Analyze ranges using derived range analysis

Timeout (minutes) ☐ Quick derived range analysis

▷ Analyze Ranges

```
1  function y = ex_2ndOrder_filter(x) %#codegen
2    persistent z
3    if isempty(z)
4        z = zeros(2,1);
5    end
6    % [b,a] = butter(2, 0.25)
7    b = [0.0976310729378175,  0.195262145875635,  0.0976310729378175];
8    a = [                 1, -0.942809041582063,  0.333333333333333];
9
10
11   y = zeros(size(x));
12   for i=1:length(x)
13       y(i) = b(1)*x(i) + z(1);
14       z(1) = b(2)*x(i) + z(2) - a(2) * y(i);
15       z(2) = b(3)*x(i)        - a(3) * y(i);
16   end
```

Variables | Function Replacements | Output

Variable	Type	Sim Min	Sim Max	Static Min	Static Max	Whole N...	Proposed Type
⊟ Input							
x	1 x 256 d...	-1	1	-1	1	No	numerictype(1, 16, 14)
⊟ Output							
y	1 x 256 d...	-0.97	1.06	-0.97	1.06	No	numerictype(1, 16, 14)
⊟ Persistent							
z	2 x 1 do...	-0.89	0.96	-0.89	0.96	No	numerictype(1, 16, 15)
⊟ Local							

Back | Next

R2016a

Version: 5.2

New Features

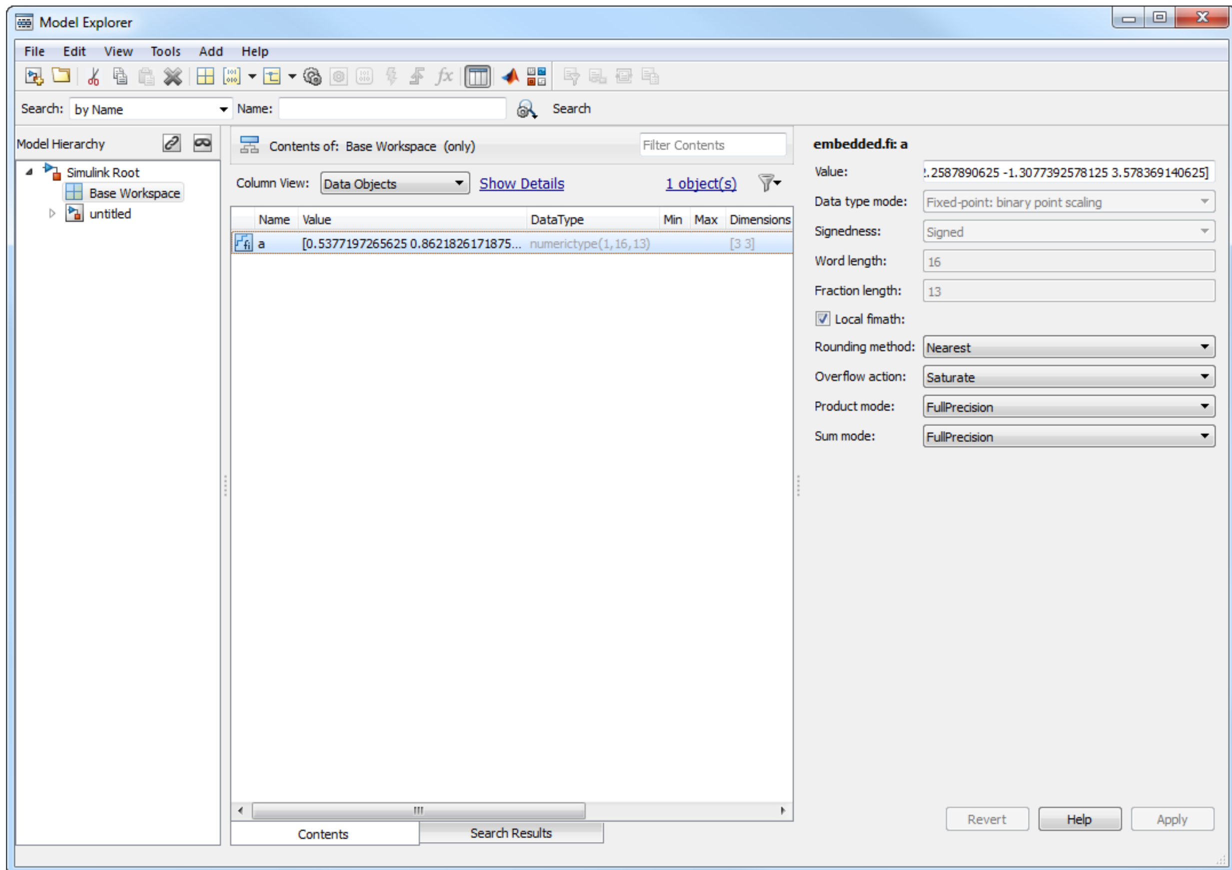
Bug Fixes

Autoscaling Parameter Objects: Automatically propose and apply data types for parameter objects

Using the Fixed-Point Tool, you can now propose and apply data types for parameter objects used in your model. The Fixed-Point Tool detects parameter objects in your model and proposes a fixed-point data type based on their respective values and ranges. The tool applies the proposed data type to the parameter object by updating the definition of the parameter object in the base or model workspace. For more information, see Autoscale Simulink.Parameter Objects.

View and edit `fi` objects in Model Explorer

You can now view and edit `fi` objects and their local `fimath` properties using Model Explorer in Simulink. You can change the writable properties of `fi` objects from the Model Explorer. You cannot change the numeric type properties of `fi` objects after creation.



Simulate system level designs that integrate referenced models targeting an assembly of heterogeneous embedded devices

When modeling larger systems, models are often composed of referenced models that target various embedded devices. You can now simulate a parent system model that includes referenced models configured with mismatching hardware settings for different embedded devices. In past releases, Simulink required the hardware settings on referenced models to match to simulate the top-level model. You can configure the hardware implementation settings in the **Configuration Parameters** > **Hardware Implementation** pane.

Enhancements to Fixed-Point Converter app

Support for arrays of structures

You can now convert arrays of structures to fixed point using the Fixed-Point Converter app. For more information on language features supported by the Fixed-Point Converter app, see MATLAB Language Features Supported for Automated Fixed-Point Conversion.

Structures in generated fixed-point code

The Fixed-Point Converter now proposes a unified data type for structures that are similar. Similar structures are structures which contain fields with the same name, number and type. The Fixed-Point Converter app no longer generates copies of structures, making the generated fixed-point code more efficient. See Convert Code Containing Structures to Fixed Point.

Revert changes to input type definitions

You can now revert and restore changes to type definitions in the **Define Input Types** step of the Fixed-Point Converter app. You can revert or restore changes in the entry-point input arguments table or the global variables table.

Use the undo and redo buttons for the table that you want to change. Alternatively, use the keyboard shortcuts for undo and redo. The keyboard shortcuts apply to the selected table. The shortcuts are defined in your MATLAB preferences. The default keyboard shortcuts for undo and redo on a Windows® platform are **Ctrl+Z** and **Ctrl+Y**.

View complete error message in error table

In previous releases, the Fixed-Point Converter app truncated a message that did not fit on one line of the error messages table on the **Convert to Fixed-Point** step. In R2016a, the app displays a long message on multiple lines so that you can see the entire message.

Additional keyboard shortcuts in the code generation report

You can now use keyboard shortcuts to perform the following actions in a code generation report.

Action	Default Keyboard Shortcut for a Windows platform
Zoom in	**Ctrl+Plus**

Action	Default Keyboard Shortcut for a Windows platform
Zoom out	**Ctrl+Minus**
Evaluate selected MATLAB code	**F9**
Open help for selected MATLAB code	**F1**
Step backward through files that you opened in the code pane	**Alt+Right**
Refresh	**F5**
Find	**Ctrl+F**

Your MATLAB preferences define the keyboard shortcuts associated with these actions. You can also select these actions from a context menu. To open the context menu, right-click anywhere in the report.

Changes to Fixed-Point Conversion Code Coverage

If you use the Fixed-Point Converter app to convert your MATLAB code to fixed-point code and propose types based on simulation ranges, the app shows code coverage results. In previous releases, the app showed the coverage as a percentage. In R2016a, the app shows the coverage as a line execution count.

```
11      persistent current_state
12      if isempty( current_state )
13          current_state = S1;                                         1 calls
14      end                                                            51 calls
15
16      % switch to new state based on the value state register
17      switch uint8( current_state )
18          case S1
19              % value of output 'Z' depends both on state and inputs
20              if (A)
21                  Z = true;                                          37 calls
22                  current_state( 1 ) = S1;
23              else                                                    7 calls
24                  Z = false;
25                  current_state( 1 ) = S2;
26              end
27          case S2                                                    51 calls
28              if (A)
29                  Z = false;                                          7 calls
30                  current_state( 1 ) = S1;
31              else                                                    0 calls
32                  Z = true;
33                  current_state( 1 ) = S2;
34              end
35          case S3                                                    51 calls
36              if (A)
37                  Z = false;                                          0 calls
38                  current_state( 1 ) = S2;
39              else
40                  Z = true;
41                  current_state( 1 ) = S3;
42              end
```

For more information, see Code Coverage.

R2015aSP1

Version: 5.0.1

Bug Fixes

R2015b

Version: 5.1

New Features

Bug Fixes

Simulink Fixed-Point Tool workflow simplification: Propose signedness and data types for inherited and floating-point types

System under design (SUD) specification

Upon opening the Fixed-Point Tool, you must now select the system under design for fixed-point conversion. Once selected, the system name will appear highlighted in green in the **Model Hierarchy** pane. The Fixed-Point Tool will propose and apply data types for the selected system only.

To change the system under design, click **Change**. In the dialog, select the system you want to convert.

Signedness proposals

The Fixed-Point Tool now proposes signedness for blocks in your system under design. To get signedness proposals for blocks in your model, in the **Automatic data typing** pane, select the **Signedness** check box.

The Fixed-Point Tool bases its signedness proposals on collected range information and block constraints. Signals that are always strictly positive now get an unsigned data type proposal, gaining an additional bit of precision compared to previous releases.

By default, the **Signedness** check box is selected. If you clear the check box, the Fixed-Point Tool proposes a signed data type for all results that currently specify a floating-point or an inherited output data type unless other constraints are present. If a result specifies a fixed-point output data type, the Fixed-Point Tool will propose a data type with the same signedness as the currently specified data type unless other constraints are present.

Proposals for objects using inherited and floating-point types

You can now elect to receive proposals for objects in your model that use floating-point data types or one of the inherited data types for block outputs. To get proposals for objects using floating-point or inherited data types, in the **Automatic data typing** pane, select the corresponding check boxes.

By default, the **Inherited** and **Floating point** check boxes are selected. If you clear the **Inherited** or **Floating point** check boxes, the Fixed-Point Tool will not propose a fixed-point data type for results that use an inherited or floating-point data type respectively.

Two-way traceability between model and Fixed-Point Tool

You can now trace between Simulink blocks in your model and their corresponding results in the Fixed-Point Tool. This capability simplifies the task of debugging overflows

and other data type propagation issues in your model. Right-click on a block in your Simulink model and select `Fixed-Point Tool Result` to highlight the result in the **Contents** pane of the Fixed-Point Tool. You can also trace a result back to the model by right-clicking a result in the **Contents** pane and selecting `Highlight in Editor`.

New configurations for model settings

Under **Configure model settings** in the Fixed-Point Tool, use the configurations to set up your model for range collection.

- The **Range collection using double override** configuration overrides the data types in your model to doubles and enables instrumentation of your model. Use these settings to collect simulation ranges using ideal floating-point data types.
- The **Range collection with specified data types** configuration removes data type override and enables instrumentation of your model. Use this shortcut to collect simulation ranges using the data types specified in your model and to validate current behavior.
- The **Remove overrides and disable range collection** configuration restores your model to its specified numeric behavior and disables instrumentation to restore maximum speed. Use this shortcut to clean up model settings after conversion.

Double-precision to single-precision conversion: Convert double-precision MATLAB code to single-precision MATLAB code using the command line

In R2015b, you can use the `convertToSingle` function to convert double-precision MATLAB code to single-precision MATLAB code.

You can verify the behavior of a single-precision version of your code without modifying the original algorithm. When a double precision operation cannot be removed, the report highlights the MATLAB expression that results in that operation.

For example, to generate single-precision MATLAB code from a double-precision function `myfunction` that takes two double arguments:

```
convertToSingle myfunction -args {1 2}
```

To use verification options, create a `coder.SingleConfig` object that you pass to `convertToSingle`. You can:

- Test numerics by running the test file with the single-precision types applied.
- Compare double-precision and single-precision test results using the Simulation Data Inspector or your own plotting functions.

```
scfg = coder.config('single');
scfg.TestBenchName = 'myfunction_test';
scfg.TestNumerics = true;
scfg.LogIOForComparisonPlotting = true;
convertToSingle -config scfg myfunction -args {1 2}
```

If you also have a MATLAB Coder™ license, you can:

- Generate single-precision C code using the MATLAB Coder app. Use this workflow if your goal is to generate single-precision C code in the most direct way and you do not want to see the intermediate single-precision MATLAB code.
- Generate single-precision C code using `codegen` with the `-singleC` option. Use this workflow when you want to generate single-precision C code in the most direct way and you do not want to see the intermediate single-precision MATLAB code
- Generate single-precision MATLAB code using `codegen` with a `coder.SingleConfig` object. Use this workflow if you want to see the single-precision MATLAB code or use verification options.
- Generate single-precision C code using `codegen` with a `coder.SingleConfig` object and a code configuration object. Use this workflow to generate single-precision C code when you also want to see the single-precision MATLAB code or use verification options.

For more information about single-precision conversion using MATLAB Coder, see the MATLAB Coder release notes.

MATLAB Fixed-Point Converter app streamlined workflow: Restore project state and minimize regeneration of MEX files

Saving and restoring fixed-point conversion workflow state in the app

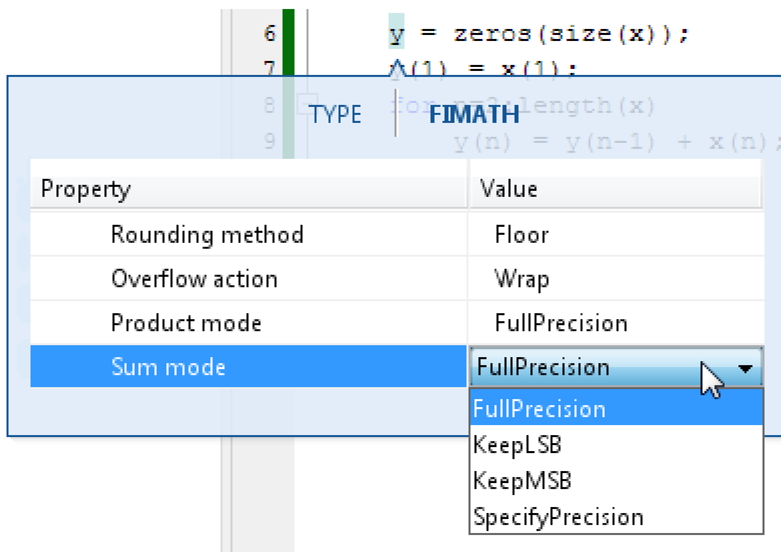
If you close a project before completing the fixed-point conversion process, the app saves your work. When you reopen the project, the app restores the state. You do not have to repeat the fixed-point conversion steps that you completed in a previous session. For example, suppose you close the project after data type proposal. When you reopen the project, the app shows the results of the data type proposal and enables conversion. You can continue where you left off.

Minimized regeneration of MEX files

The Fixed-Point Converter app now optimizes when it regenerates MEX files. The app will only rebuild the MEX file when required by changes in your code.

Specification of additional `fimath` properties in app editor

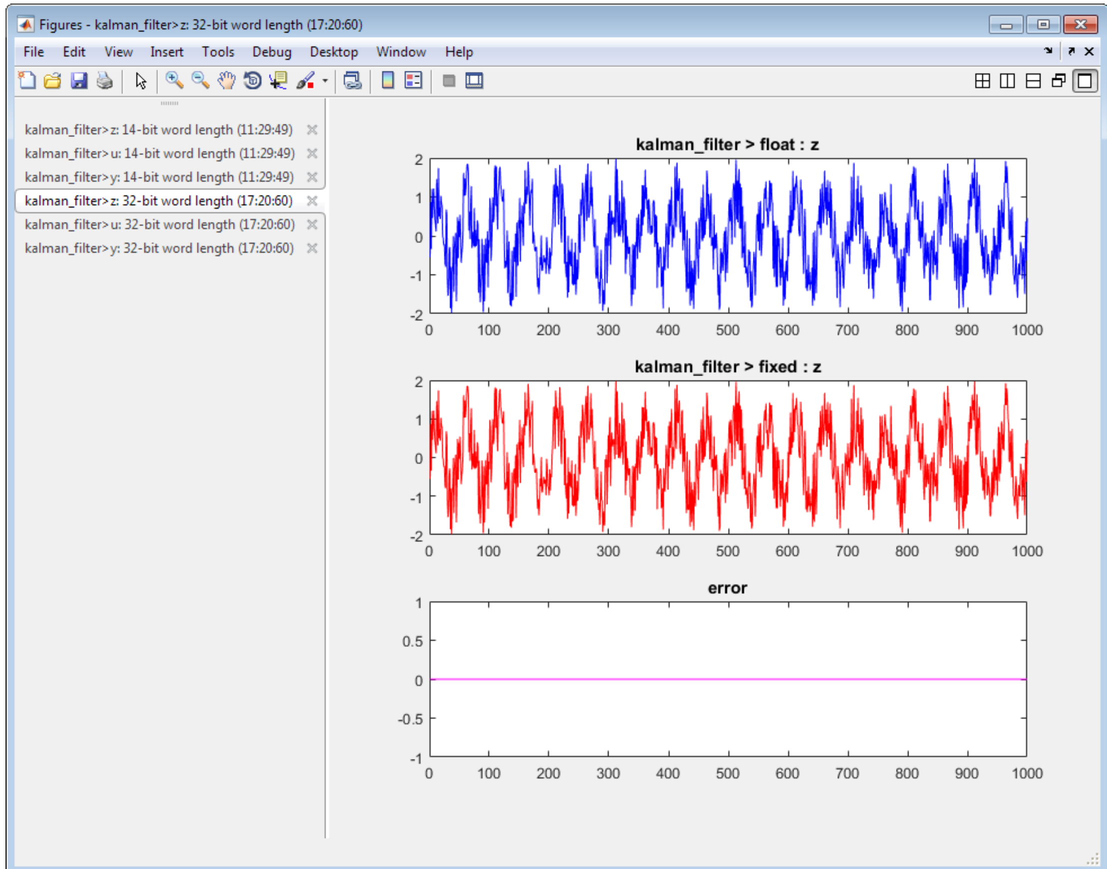
You can now control all `fimath` properties of variables in your code from within the Fixed-Point Converter app editor. To modify the `fimath` settings of a variable, select a variable and click **FIMATH** in the dialog that appears. You can alter the Rounding method, Overflow action, Product mode, and Sum mode properties. You can also modify these properties from the settings pane. For more information on these properties, see fimath.



Improved management of comparison plots

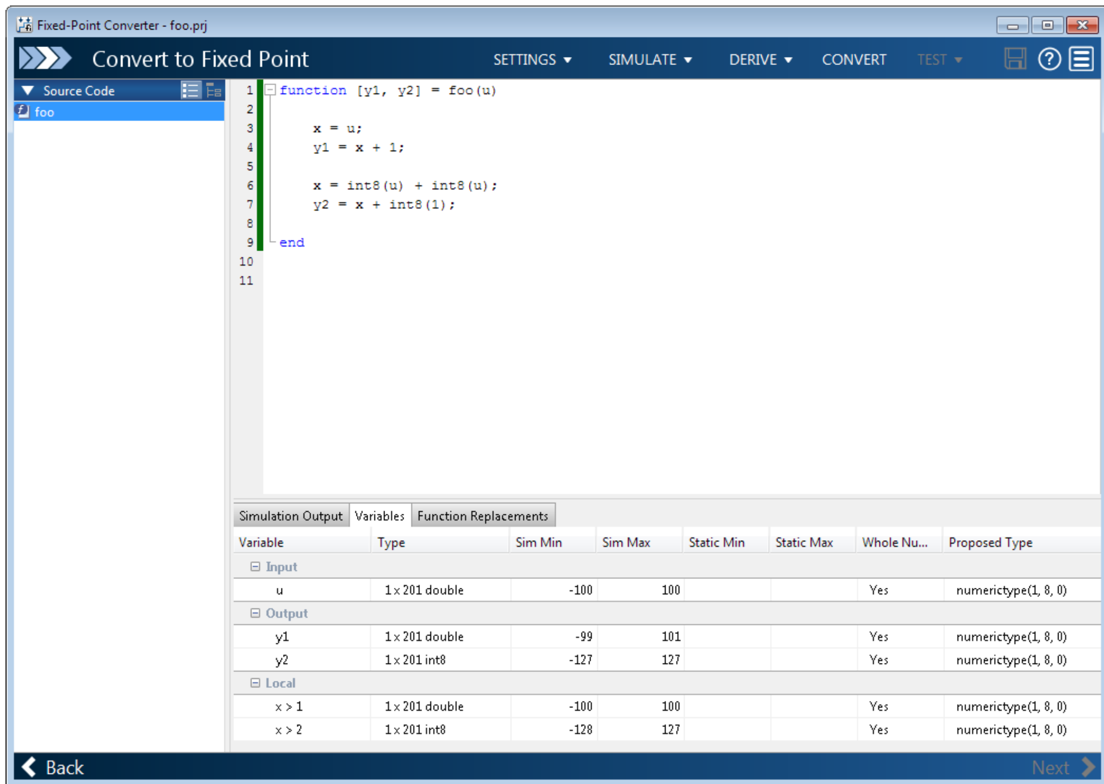
The Fixed-Point Converter app now docks plots generated during the testing phase of your fixed-point code into separate tabs of one figure window. Each tabbed figure represents one input or output variable and is labeled with the function, variable, word length, and a timestamp. Each tab contains three sub plots. The plots use a time series based plotting function to show the floating-point and fixed-point results and the difference between them.

Subsequent iterations are also plotted in the same figure window.



Variable specializations

On the **Convert to Fixed Point** page, in the **Variables** table of the app, you can now view variable specializations.



Improvements to Readability of Generated Code

Structs

- When struct copies exist in the design, a separate function is now created to perform the copy.
- Copies of structs are now avoided when the types of all fields match, improving both readability and efficiency of the generated code.

fimath

- fimath settings are now specified in a separate function to improve the readability of the generated fixed-point code.
- To avoid a mismatch of `fimath` settings in an expression, the generated code now uses the removefimath function.

```
function [y] = my_add_fixpt(a,b)
%Adds a and b
fm = getConversionFimath();

y=fi(removefimath(a)+b, 0, 8, 0, fm);
end

function fm = getConversionFimath()
 fm = fimath('RoundingMethod', 'Floor', 'OverflowAction', 'Wrap',...
 'ProductMode', 'FullPrecision', 'MaxProductWordLength', 128,...
 'SumMode', 'FullPrecision', 'MaxSumWordLength', 128);
end
```

Matrices

Growth and deletion of matrices within a design are now supported for fixed-point conversion.

```
function matrix_deletion_fixpt(a,i)
    fm = getConversionFimath();

    var = fi([1, 2, 3], 0, 2, 0, fm);
    coder.varsize('var');
    var(2) = []; % matrix deletion.
    var(2) = fi(2, 0, 2, 0, fm);
end

function [out] = matrix_growth_fixpt( x )
    fm = getConversionFimath();
    out = fi([], 0, 4, 0, fm);
    for ii = 1:10
        out = [ out x];
    end
end
```

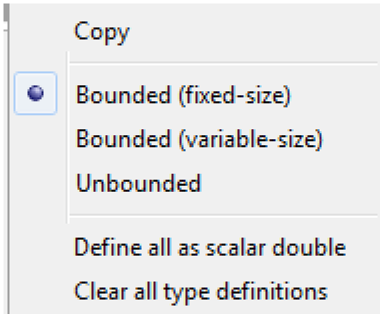
Tab completion for specifying files

On the **Select Source Files** and **Define Input Types** pages of the Fixed-Point Converter app, you can now use tab completion to specify your entry-point functions and test bench file.

Improvements for manual type definition

Improvements for manual type definition include:

- New right-click menus options to specify array size.



- Easier definition of structure types. You can:
  - Use the new + icon to add fields.
  - See the structure type name in the table of input variables.

⊟ x	struct(1 x 2)	*myname*
field1	double(1 x 1)	

- Easier definition of `embedded.fi` types. You can:
  - See the `numerictype` properties in the table of input variables.

x	fi(1 x 1)	*numerictype(1, 16, 15)*

  - Use the new icon to change the `numerictype` properties.

Compatibility between the app colors and MATLAB preferences

The app uses colors that are compatible with the **Desktop tool colors** preference in the MATLAB preferences. For information about MATLAB preferences, see Preferences.

Range analysis for Delay blocks: Improve accuracy and speed of range analysis on models using Delay blocks

Using the Fixed-Point Tool, you can now derive ranges for models that use Delay blocks with greater precision. The Fixed-Point Tool can also derive ranges for certain

configurations of cascading Delay blocks with greater theoretical accuracy and speed. For more information on range analysis in the Fixed-Point Tool, see How Range Analysis Works.

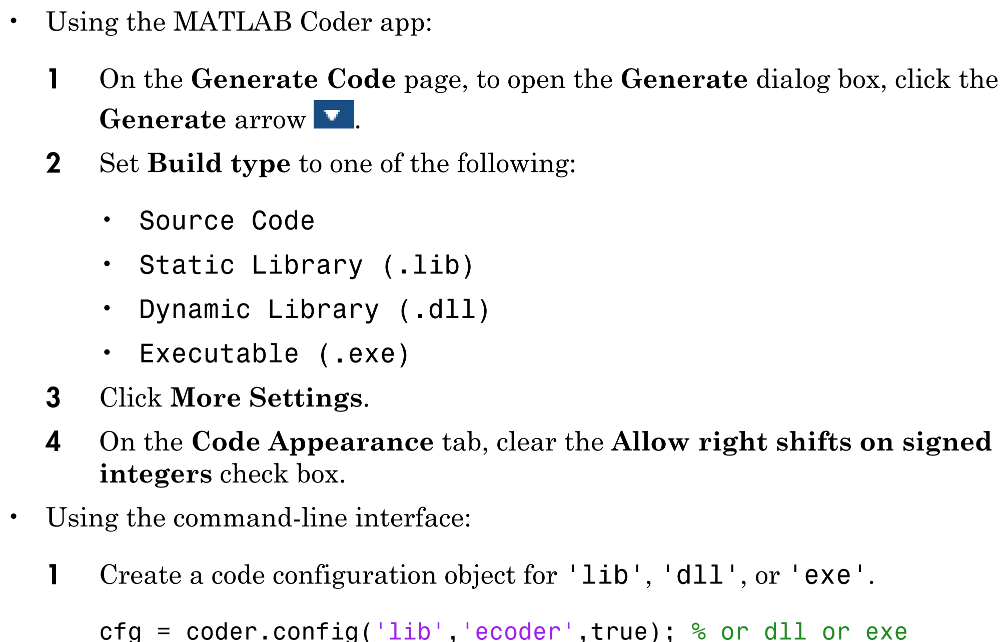
Control of signed shifts in fixed-point scaling operations: Control the use of signed shifts in generated code

You can now control the use of signed right shifts in your generated code. Some coding standards do not allow bitwise operations on signed integers. Disabling the use of signed shifts in generated code increases the likelihood of compliance with MISRA. When you specify that signed right shifts should not be used in your generated code, the software replaces signed shifts with a call to a function that performs the operation without the use of signed shifts.

This feature requires an Embedded Coder® license.

MATLAB

To specify that MATLAB Coder not use signed right shifts:



- Using the MATLAB Coder app:
  1. On the **Generate Code** page, to open the **Generate** dialog box, click the **Generate** arrow.
  2. Set **Build type** to one of the following:
     - `Source Code`
     - `Static Library (.lib)`
     - `Dynamic Library (.dll)`
     - `Executable (.exe)`
  3. Click **More Settings**.
  4. On the **Code Appearance** tab, clear the **Allow right shifts on signed integers** check box.
- Using the command-line interface:
  1. Create a code configuration object for `'lib'`, `'dll'`, or `'exe'`.

     ```
     cfg = coder.config('lib','ecoder',true); % or dll or exe
     ```

2 Set the `EnableSignedRightShifts` property to `false`.

```
cfg.EnableSignedRightShifts = false;
```

Simulink

To specify that the code generator not use signed right shifts, in the Configuration Parameters dialog box, on the **Code Generation** > **Code Style** pane, clear Allow right shifts on signed integers or set the parameter `EnableSignedRightShifts` to `off`.

To improve coding standard compliance for bitwise operations on signed integers, run the following checks:

- Check for bitwise operations on signed integers - Check to identify blocks that contain bitwise operations on signed integers.
- Check configuration parameters for MISRA C:2012 - Check that verifies that you cleared **Code Generation** > **Code Style** > **Allow right shifts on signed integers**.

Access full-precision value of `fi` object in decimal and string format

You can now set and get full-precision real-world values of fi objects using the new `Value` property. This provides easy access to exact values in decimal format.

The tostring function now accepts `fi` object inputs allowing you to convert `fi` objects to a string that you can copy and paste into a MATLAB script or function. The mat2str function now also supports `fi` object inputs allowing you to convert `fi` objects to strings without first converting to a double value.

Detection of multiword operations

When an operation has an input or output larger than the largest word size of your processor, the generated code contains multiword operations. Multiword operations can be inefficient on hardware. In both MATLAB and Simulink, you can now detect operations that will result in multiword code.

MATLAB

The expensive fixed-point operations check now highlights expressions in your MATLAB code that could result in multiword operations in generated code. For more information on enabling this check, see Find and Address Multiword Operations.

Simulink

The Identify questionable fixed-point operations check in the Model Advisor now detects multiword operations in generated code. For more information, see Identify Questionable Fixed-Point Operations.

Enhanced Model Advisor check for implementing strict single-precision designs

The Model Advisor **Modeling Single-Precision Systems** > **Identify questionable operations for strict single-precision design** check now verifies the status of additional model settings that will help you achieve a strict single-precision design.

- The Model Advisor warns you if **Configuration Parameters** > **Optimization** > **Default for underspecified data type** is set to `Double`.
- The Model Advisor warns you if your model uses library standard that is not optimal for strict-single precision designs.
- The Model Advisor warns you if **Configuration Parameters** > **Optimization** > **Implement logic signals as Boolean data** is not selected.

  The settings suggested by the Model Advisor prevent the introduction of doubles into your generated code, which is optimal for strict-single designs.

System object instrumentation in Fixed-Point Tool

The Fixed-Point Tool now collects simulation ranges and proposes data types for select DSP System Toolbox™ System objects used inside a MATLAB Function block. You cannot propose data types based on derived range data.

Use of these System objects requires a DSP System Toolbox license. To learn more about using the Fixed-Point Tool to convert System objects and to learn which System objects are supported, see Convert a System Object to Fixed Point Using the Fixed-Point Tool.

R2015a

Version: 5.0

New Features

Bug Fixes

Derived Ranges for MATLAB Function Blocks in Simulink

Using the Fixed-Point Tool, you can now derive ranges for variables inside a MATLAB Function block in Simulink. The Fixed-Point Tool uses design ranges to derive ranges for MATLAB variables in a MATLAB Function block. The tool can also propose data types for the variables based on the derived range data. You must manually apply the proposed data types to the variables. For more information, see Derive Ranges of MATLAB Function Block Variables.

Fixed-Point Converter app enhancements, including detection of dead and constant folded code, support for projects with multiple entry point functions and support for global variables

The following enhancements have been added to the Fixed-Point Converter app:

Support for projects with multiple entry-point functions

You can now specify multiple entry-point functions in a Fixed-Point Converter app project. If your end goal is to generate fixed-point C/C++ library functions, conversion with multiple entry-point functions facilitates integration with larger applications. For more information, see Generate Fixed-Point MATLAB Code for Multiple Entry-Point Functions.

Support for global variables

You can now specify global variables in the Fixed-Point Converter app workflow and convert algorithms which contain global variables without modifying your code. For more information, see Convert Code Containing Global Variables to Fixed-Point.

Code coverage based translation

The Fixed-Point Converter app now detects dead and constant folded code within your project and warns you if any parts of your code were not executed during the simulation of your test file. This can help you verify if your test file is testing the algorithm over the intended operating range. The app uses this code coverage information during the translation of your code from floating-point MATLAB code to fixed-point MATLAB code. The app inserts inline comments in the fixed-point code to mark the dead and untranslated regions and includes the code coverage information in the generated fixed-point conversion html report. This code coverage information is also available from the

command-line workflow. For more information, see Detect Dead and Constant-Folded Code.

Conversion from project to MATLAB scripts for command-line fixed-point conversion

Using the `-tocode` option of the `fixedPointConverter` command, you can convert a fixed-point conversion project to the equivalent MATLAB code in a MATLAB script. You can use the script to repeat the project workflow in a command-line workflow. For more information, see Convert Fixed-Point Conversion Project to MATLAB Scripts.

Generated fixed-point code enhancements

The generated fixed-point code now:

- Uses colon syntax for multi-output assignments, reducing the number of `fi` casts in the generated fixed-point code.
- Preserves the indentation and formatting of your original algorithm, improving the readability of the generated fixed-point code.

Integration with MATLAB Coder app interface

The Fixed-Point Converter app has been integrated into the new MATLAB Coder app workflow. This integration allows for a smoother conversion process from floating-point MATLAB code to fixed-point C/C++ code.

Automated conversion of additional DSP System objects using the Fixed-Point Converter app

You can now convert the following DSP System Toolbox System objects to fixed-point using the Fixed-Point Converter app:

- dsp.FIRDecimator
- dsp.FIRInterpolator
- dsp.FIRFilter, direct form and direct form transposed only
- dsp.LUFactor
- dsp.VariableFractionalDelay
- dsp.Window

You can propose and apply data types for these System objects based on simulation range data. During the conversion process, you can view simulation minimum and

maximum values and proposed data types for these System objects. You can also view whole number information and histogram data. You cannot propose data types for these System objects based on static range data. This requires a DSP System Toolbox license.

Fixed-Point SimState logging and root logging improvements

The Simulink `SimState` feature allows you to save all run-time data necessary for restoring the simulation state of the model. A `SimState` includes both the logged and internal state of every block and the internal state of the Simulink engine. The Fixed-Point Tool now supports `SimState` logging while fixed-point instrumentation is turned on. For more information, see Save and Restore Simulation State as SimState.

Flexible structure assignment of buses

When a non-tunable structure is assigned to a bus signal (such as a block which uses a structure for its initial condition parameter), the data type of the fields of the structure no longer need to match the data type of the bus elements. The software now performs an automatic casting of the data type of the structure field so that it matches the da;ta type of the bus signal. This flexible structure assignment simplifies the fixed-point conversion workflow by automatically casting the data type of the fields of the structure when using data type override and autoscaling your model.

`eye(m,'like',a)` syntax supported for fixed-point inputs

The eye function now works with fixed-point data types as well as built-in data types. The function can now return an output whose class matches that of a specified numeric variable or `fi` object. For built-in data types, the output assumes the numeric data type, sparsity, and complexity (real or complex) of the specified numeric variable. For `fi` objects, the output assumes the `numerictype`, complexity (real or complex), and `fimath` of the specified `fi` object.

New interpolation method for generating lookup table MATLAB function replacements

The coder.approximation function now offers a `'Flat'` interpolation method for generating lookup table MATLAB function replacements. This fully-specified lookup table achieves high speeds by discarding the pre-lookup step and reducing the use of multipliers in the data path. This interpolation method is available from both the

command-line workflow, and in the **Function Replacements** tab of the Fixed-Point Converter app.

Fixed-point scaling information in Code Interface Report

Fixed-point scaling information is added to the code generation report in the Code Interface Report section. Better accessibility to this information makes it easier for you to integrate with generated code containing fixed-point data types. Each fixed-point entry in the report table has a value in the new **Scaling** column giving its data type and fraction length using Simulink fixed-point data type notation.

Access to the Code Interface Report requires an Embedded Coder license.

R2014b

Version: 4.3

New Features

Bug Fixes

Compatibility Considerations

Fixed-Point Converter app for automated conversion of floating-point MATLAB code

The Fixed-Point Converter app enables you to convert floating-point MATLAB code to fixed-point MATLAB code.

You can choose to propose data types based on simulation range data, static range data, or both.

During fixed-point conversion, you can:

- Propose fraction lengths based on default word lengths.
- Propose word lengths based on default fraction lengths.
- Optimize whole numbers.
- Specify safety margins for simulation min/max data.
- Test numerics by running the test file with the fixed-point types applied.
- Compare floating-point and fixed-point test results using the Simulation Data Inspector or your own plotting functions.
- View a histogram of bits used by each variable.
- Specify replacement functions and generate approximate functions for functions in the original MATLAB algorithm that are not supported for fixed point.

To open the app:

- In the MATLAB Toolstrip, on the **Apps** tab, under **Code Generation**, click .
- At the MATLAB command prompt, enter `fixedPointConverter`.

For more information, see Fixed-Point Converter.

Commands for scripting fixed-point conversion and accessing the collected data in Simulink

You can now use the DataTypeWorkflow.Converter class to collect simulation and derived data, propose and apply data types to the model, and analyze results.

This class performs the same fixed-point conversion tasks as the Fixed-Point Tool. This facilitates scripting of the automatic conversion workflow and accessing data for analysis. For more information, see Convert a Model to Fixed Point Using the Command-Line.

Automated fixed-point conversion for commonly used DSP System objects, including Biquad Filter, FIR Filter, and FIR Rate Converter

You can now convert the following DSP System Toolbox System objects to fixed point using the Fixed-Point Converter app.

- dsp.BiquadFilter
- dsp.FIRFilter, direct form only
- dsp.FIRRateConverter
- dsp.LowerTriangularSolver
- dsp.UpperTriangularSolver
- dsp.ArrayVectorAdder

You can propose and apply data types for these System objects based on simulation range data. During the conversion process, you can view simulation minimum and maximum values and proposed data types for these System objects. You can also view whole number information and histogram data. You cannot propose data types for these System objects based on static range data. This requires a DSP System Toolbox license. For more information, see Convert a System object to Fixed-Point Using the Fixed-Point Converter App.

Simulation range collection and data type proposals for MATLAB Function blocks in Simulink

The Fixed-Point Tool can now collect and display simulation ranges for variables inside a MATLAB Function block. The tool can also propose data types for the variables based on the simulation data. You must manually apply the proposed data types to the variables. For more information, see Convert Model Containing MATLAB Function Block to Fixed Point.

Overflow diagnostics to distinguish between wrap and saturation in Simulink

You can now separately control the diagnostics for overflows that wrap and overflows that saturate by setting each diagnostic to `error`, `warning`, or `none`. These controls simplify debugging models in which only one type overflow is of interest. For example, if you need to detect only overflows that wrap, in the **Data Validity** pane of the

Configuration Parameters dialog box you can set **Wrap on overflow** to `error` or `warning`, and set **Saturate on overflow** to `none`.

Highlighting of potential data type issues in generated HTML report

You can now highlight potential data type issues in the generated HTML report. The report highlights MATLAB code that requires single-precision, double-precision, or expensive fixed-point operations. The expensive fixed-point operations check identifies optimization opportunities by highlighting expressions in the MATLAB code that require cumbersome multiplication or division, or expensive rounding.

For more information, see Find Potential Data Type Issues in Generated Code

Code generation of for loops using fixed-point loop indices

Fixed-point data types are now supported as for-loop indices in `codegen`. This capability requires a MATLAB Coder license. For more information, see for.

Cast net slope computations using rational numbers

This new option improves the numerical accuracy and the readability of the C code generated for certain fixed-point conversions having nonbinary net slopes. Normally, net slope computation uses an integer multiplication followed by shifts. Enabling this optimization replaces the multiply and shift operation with a multiply and divide sequence that uses a rational number under certain simplicity and accuracy conditions.

For example, applying a net slope of 0.9, which traditionally would have generated

```
Vc = (int16_T)(Va * 115 >> 7);
```

becomes

```
Vc = (int16_T)(Va * 9/10);
```

This optimization affects both simulation and code generation. For more information, see Handle Net Slope Computation.

Lock Column View option in the Fixed-Point Tool

This option prevents the Fixed-Point Tool from automatically changing the column view of the contents pane. To enable this option, in the Fixed-Point Tool menu, click **View** > **Lock Column View**. This setting is preserved across sessions.

Fixed-Point Advisor enhancements

- Improved support for interaction with Simulink data objects, including bus objects
- Block replacement recommendations for blocks with CORDIC support

hdlram renamed hdl.RAM

The `hdlram` System object™ has been renamed `hdl.RAM`. This System object no longer requires a Fixed-Point Designer™ license.

Compatibility Considerations

If you open a design that uses `hdlram`, the software displays a warning. For continued compatibility with future releases, replace instances of `hdlram` with `hdl.RAM`.

Changes to data type strings

Signal data type display

Signals using fixed-point data types with slope and bias scaling now always display the slope value in the data type name. In previous releases, the display decomposed the slope into slope adjustment factor and fixed exponent when it led to a more compact string. For example, the data type `fixdt(1,32,0.01953125,0)` now gets the name `sfix32_S0p01953125`. In previous releases, the name was in the decomposed format `sfix32_F1p25_en6`.

tostring function now uses 0 and 1 to represent signedness

The string representation of `numerictype` and `fixdt` objects returned by the tostring function now use `0` and `1` to represent signedness rather than `true` and `false`.

```
T = numerictype(true,16,15);
T.tostring

ans =

numerictype(1,16,15)
```

When programmatically processing data types, best practice is to convert string representations to `numerictype` objects. The string changes for this release do not change the object that the strings are converted to. To convert a data type name string

to an object, pass the string as the input argument to `fixdt` or `numerictype`. For example, `fixdt('sfix32_S0p01953125')` and `fixdt('sfix32_F1p25_En6')` return identical `numerictype` objects. To convert the results of the `tostring` function back to an object, use the eval function. For example, the `numerictype` objects returned by `eval('numerictype(1,16,15)')` and `eval('numerictype(true,16,15)')` are identical.

Compatibility Considerations

If your code converts data type strings to objects before doing any processing, then you will not have any compatibility issues related to the string changes. If you depend on the exact text returned by the `tostring` function or the exact text of a Simulink data type name, then you must modify your code to account for the changes described here. Alternatively, you can convert the string to a `numerictype` object before doing any additional processing.

New featured examples

The Fixed-Point Conversion Using Fixed-Point Tool and Derived Range Analysis example demonstrates using derived range analysis and the Fixed-Point Tool to convert a corner detection model to fixed point.

R2014a

Version: 4.2

New Features

Bug Fixes

Data type override and automatic data typing for bus objects

Data type override for bus objects

You can now apply data type override to models and subsystems that use virtual and non-virtual buses. The bus element types obey the data type override settings. This capability allows you to:

- Obtain the idealized floating-point behavior of models that use buses.
- Obtain the ideal derived ranges for models that use buses.
- Easily compare the idealized floating-point behavior with the fixed-point behavior of models that use buses.
- Use data type override to share fixed-point models that use buses with users who do not have a fixed-point license.

Autoscaling for bus objects

You can autoscale models that use virtual and non-virtual buses. This capability facilitates fixed-point conversion and optimization of models. The Fixed-Point Tool automatically proposes fixed-point data types for bus elements which removes the need to perform manual analysis and conversion of bus element data types.

For more information, see Refine Data Types of a Model with Buses Using Simulation Data.

Derived ranges for complex signals in Simulink

Using the Fixed-Point Tool, you can now derive ranges for complex signals in Simulink. For more information, see Conversion Using Range Analysis.

`cordicsqrt` function for fixed-point CORDIC-based square root functionality

The `cordicsqrt` function provides a CORDIC-based approximation of square root for use in fixed-point applications. For more information, see cordicsqrt and Compute Square Root Using CORDIC.

Overflow detection with scaled double data types in MATLAB Coder projects

The MATLAB Coder Fixed-Point Conversion tool now provides the capability to detect overflows. At the numerical testing stage in the conversion process, the tool simulates the fixed-point code using scaled doubles. It then reports which expressions in the generated code produce values that would overflow the fixed-point data type. For more information, see Detect Overflows Using the Fixed-Point Conversion Tool and Detecting Overflows.

You can also detect overflows when using the `codegen` function. For more information, see coder.FixptConfig and Detect Overflows at the Command Line.

These capabilities require a MATLAB Coder license.

Fixed-point ARM Cortex-M code replacement support for DSP System Toolbox FIR filters

Fixed-point ARM® Cortex®-M code replacement library support is now available for the Discrete FIR block and the dsp.FIRFilter System object.

These capabilities require a DSP System Toolbox license.

Fixed-Point Advisor support for referenced configuration sets

The Fixed-Point Advisor now supports referenced configuration sets. For more information, see Preparing for Data Typing and Scaling.

Enhancements to automated conversion of MATLAB code

R2014a includes the following enhancements to the fixed-point conversion capability in MATLAB Coder projects.

These capabilities require a MATLAB Coder license.

Support for MATLAB classes

You can now use the MATLAB Coder Fixed-Point Conversion tool to convert floating-point MATLAB code that uses MATLAB classes. For more information, see Fixed-Point Code for MATLAB Classes.

Generated fixed-point code enhancements

The generated fixed-point code now:

- Uses subscripted assignment (the colon(:) operator). This enhancement produces concise code that is more readable.
- Has better code for constant expressions. In previous releases, multiple parts of an expression were quantized to fixed point. The final value of the expression was less accurate and the code was less readable. Now, constant expressions are quantized only once at the end of the evaluation. This new behavior results in more accurate results and more readable code.

For more informations, see Generated Fixed-Point Code.

Fixed-point report

In R2014a, when you convert floating-point MATLAB code to fixed-point C/C++ code, the code generation software generates a fixed-point report in HTML format. For the variables in your MATLAB code, the report provides the proposed fixed-point types and the simulation or derived ranges used to propose those types. For a function, `my_fcn`, and code generation output folder, `out_folder`, the location of the report is `out_folder/my_fcn/fixpt/my_fcn_fixpt_Report.html`. If you do not specify `out_folder` in the project settings or as an option of the `codegen` command, the default output folder is `codegen`.

Automatic C compiler setup

In earlier releases, to set up a compiler before using fiaccel to accelerate MATLAB algorithms, you were required to run `mex -setup`. Now, the code generation software automatically locates and uses a supported installed compiler. You can use `mex -setup` to change the default compiler. See Changing Default Compiler.

More flexible control of dsp.LMSFilter System object fixed-point settings

For all dsp.LMSFilter System object fixed-point settings, you can now specify independent fixed-point data types.

This capability requires a DSP System Toolbox license.

Derived ranges for For Each and For Each Subsystem blocks

Range analysis supports For Each and For Each Subsystem blocks, with the following limitations:

- When For Each Subsystem contains another For Each Subsystem, not supported.
- When For Each Subsystem contains one or more Simulink Design Verifier™ Test Condition, Test Objective, Proof Assumption, or Proof Objective blocks, not supported.

R2013b

Version: 4.1

New Features

Bug Fixes

Compatibility Considerations

C99 long long integer data type for embedded code generation

If your target hardware and your compiler support the C99 long long integer data type, you can use this data type for code generation. Using long long results in more efficient generated code that contains fewer cumbersome operations. Multi-line fixed-point helper functions can be replaced by simple expressions. This data type also provides more accurate simulation results for fixed-point and integer simulations. If you are using Microsoft® Windows (64-bit), using long long improves performance for many workflows including:

- Using Accelerator mode in Simulink
- Working with Stateflow® software
- Generating C code with Simulink Coder
- Accelerating fixed-point code using `fiaccel`
- Generating C code and MEX functions with MATLAB Coder

For more information about enabling long long in Simulink, see the **Enable long long** and **Number of bits: long long** configuration parameters on the Hardware Implementation Pane.

For more information about enabling long long for MATLAB Coder, see coder.HardwareImplementation.

Model Advisor fixed-point checks with additional coverage and optimization awareness

The Model Advisor fixed-point checks now cover additional blocks in base Simulink and System Toolboxes. The checks also now include the MATLAB Function block, System objects, Stateflow, and `fi` objects. These improved checks consider model settings such as hardware configuration and code generation settings. These updated checks also avoid false negative results.

These checks require an Embedded Coder license.

For more information, see:

- Identify blocks that generate expensive rounding code
- Identify questionable fixed-point operations
- Identify blocks that generate expensive fixed-point and saturation code

`fi` object as an index in colon expressions and an argument to `numel` and bit index functions

fi object as an index in colon expressions

You can now use `fi` objects in colon expressions. When you use `fi` in a colon expression, all colon operands must have integer values. See the fi and colon reference pages for examples.

fi objects as bit index input argument

The bitget, bitset, bitsliceget, bitandreduce, bitorreduce, and bitxorreduce functions now accept `fi` objects as the bit index argument.

fi objects as shift-value input argument

The bitsra, bitsrl, bitsll, bitrol, and bitror functions now accept `fi` objects as the shift-value input argument. You can use `fi` and built-in data type shift values interchangeably in MATLAB functions. This new capability facilitates fixed-point conversion.

numel function support for fi inputs

Effective R2013b, the `numel` function returns the number of elements in a `fi` array. Using `numel` in your MATLAB code returns the same result for built-in types and `fi` objects. Use `numel` to write data-type independent MATLAB code for array handling; you no longer need to use the `numberofelements` function.

The `numel` function is supported for simulation and code generation and with the MATLAB Function block in Simulink.

For more information, see numel.

Improved efficiency of data type internal rules for Lookup Table blocks

Blocks in the Lookup Tables library have a new internal rule for fixed-point data types to enable faster hardware instructions for intermediate calculations (with the exception of the Direct Lookup Table (n-D), Prelookup and Lookup Table Dynamic blocks). To use this new rule, select `Speed` for the **Internal Rule Priority** parameter in the dialog box. To use the R2013a internal rule, select `Precision`.

Derived ranges for complex variables in MATLAB Coder projects

Using the Fixed-Point Conversion tool in MATLAB Coder projects, you can now derive ranges for complex variables. For more information, see Propose Data Types Based on Derived Ranges. This capability requires a MATLAB Coder license.

Simplified modeling of single-precision designs

Fixed-Point Designer now uses strict single-precision algorithms for operations between singles and integer or fixed-point data types. Operations, such as cast, multiplication and division, use single-precision math instead of introducing higher-precision doubles for intermediate calculations in simulation and code generation. You no longer have to explicitly cast integer or fixed-point inputs of these operations to single precision. To detect the presence of double data types in your model, use the Model Advisor Identify questionable operations for strict single-precision design check.

Compatibility Considerations

In R2013b, for both simulation and code generation, Fixed-Point Designer avoids the use of double data types to achieve strict single design for operations between singles and integers or fixed-point types. In previous releases, Fixed-Point Designer used double data types in intermediate calculations for higher precision. You might see a difference in numerical behavior of an operation between earlier releases and R2013b.

For example, when you cast from a fixed-point or integer data type to single or vice versa, the type used for intermediate calculations can significantly affect numerical results. Consider:

- Input type: `ufix128_En127`
- Input value: 1.999999999254942 — Stored integer value is `(2^128 -2^100)`.
- Output type: single

Release	Calculation performed by Fixed-Point Designer	Output Result	Design Goal
R2013b	`Y = single(2^-127) * single(2^128-2^100)` `= single(2^-127) * Inf`	Inf	Strict singles
Previous releases	`Y = single( double(2^-127) * double(2^128 - 2^100))` `= single(2^-127 * 3.402823656532e+38)`	2	Higher-precision intermediate calculation

There is also a difference in the generated code. Previously, Fixed-Point Designer allowed the use of doubles in the generated code for a mixed multiplication that used single and integer types.

```
m_Y.Out1 = (real32_T)((real_T)m_U.In1*(real_T)m_U.In2);
```

In R2013b, it uses strict singles.

```
m_Y.Out1=(real32_T)m_U.In1*m_U.In2;
```

You can revert to the numerical behavior of previous releases, if necessary. To do so, insert explicit casting from integer and fixed-point data types to doubles for the inputs of these operations.

Range analysis support on Mac platforms

You can now perform derived range analysis of your model on Mac platforms. For more information, see Conversion Using Range Analysis.

Changes to showInstrumentationResults function options

New option to suppress display of MATLAB code

When generating a printable instrumentation report, you can now choose to display only the tables that show information about logged variables. Used with the `-printable` option, the `-nocode` option suppresses display of the MATLAB code. Displaying only the logged variable information is useful for large projects with many lines of code.

Removal of `-browser` option

The `showInstrumentationResults` function `-browser` option has been removed. Use the `-printable` option instead. The `-printable` option creates a printable report and opens it in the system browser.

For more information, see showInstrumentationResults.

Changes to Continuous state-space block family range analysis support

The Continuous Simulink blocks State-Space, Transfer Fcn, and Zero-Pole are not supported and not stubbable for range analysis. For more information on blocks that are supported for range analysis, see Supported and Unsupported Simulink Blocks.

Compatibility Considerations

If you have a model that contains one or more continuous State-Space, Transfer Fcn, or Zero-Pole blocks, your model is incompatible with range analysis. Consider analyzing smaller portions of your model to work around this incompatibility.

Enhanced `fiaccel` support for `int64` and `uint64` functions

The `fiaccel` function now supports int64 and uint64 with `fi` inputs.

Support for LCC compiler on Microsoft Windows (64-bit) machines

If you are using Microsoft Windows (64-bit), LCC-64 is now available as the default compiler. You no longer have to install a separate compiler to perform fixed-point acceleration using fiaccel.

Warning for use of inexact `fi` and `fimath` property names

All `fi` and `fimath` property names are case sensitive and require that you use the full property names. Effective R2013b, if you use inexact property names, Fixed-Point Designer generates a warning.

Compatibility Considerations

To avoid seeing warnings for `fi` and `fimath` properties, update your code so that it uses the full names and correct cases of all these properties. The full names and correct cases of the properties appear when you display a `fi` or `fimath` object on the MATLAB command line.

Conversion of numeric variables into `Simulink.Parameter` objects

You can now convert a numeric variable into a `Simulink.Parameter` object using a single step.

```
myVar = 5; % Define numerical variable in base workspace
myObject = Simulink.Parameter(myVar); % Create data object and assign variable value to data object value
```

Previously, you did this conversion using two steps.

```
myVar = 5; % Define numerical variable in base workspace
myObject = Simulink.Parameter; % Create data object
```

```
myObject.Value = myVar; % Assign variable value to data object value
```

Fixed-point conversion test file coverage results

The MATLAB Coder Fixed-Point Conversion tool now provides test file coverage results. After simulating your design using a test file, the tool provides an indication of how often the code is executed. If you run multiple test files at once, the tool provides the cumulative coverage. This information helps you determine the completeness of your test files and verify that they are exercising the full operating range of your algorithm. The completeness of the test file directly affects the quality of the proposed fixed-point types.

This capability requires a MATLAB Coder license.

For more information, see Code Coverage.

Fixed-point conversion workflow supports designs that use enumerated types

Using the Fixed-Point Conversion tool in MATLAB Coder projects, you can now propose data types for enumerated data types using derived and simulation ranges.

For more information, see Propose Fixed-Point Data Types Based on Derived Ranges and Propose Fixed-Point Data Types Based on Simulation Ranges. This capability requires a MATLAB Coder license.

Fixed-point conversion of variable-size data using simulation ranges

Using the Fixed-Point Conversion tool in MATLAB Coder projects, you can propose data types for variable-size data using simulation ranges.

For more information, see Propose Fixed-Point Data Types Based on Simulation Ranges. This capability requires a MATLAB Coder license.

Error checking improvements for `bitconcat`, `bitandreduce`, `bitorreduce`, `bitxorreduce`, `bitsliceget` functions

The bitconcat, bitandreduce, bitorreduce, bitxorreduce, and bitsliceget functions now check that all input arguments are real. If any inputs are complex, these functions generate an error.

The `bitconcat` function now generates an error in the unary syntax case, `bitconcat(a)`, if the input argument `a` is a scalar or is empty. To use `bitconcat` with one input argument, the argument must have more than one array element available for bit concatenation (that is, `length(a)>1`).

Legacy data type specification functions return numeric objects

In previous releases, the following functions returned a MATLAB structure describing a fixed-point data type:

- float
- sfix
- sfrac
- sint
- ufix
- ufrac
- uint

Effective R2013b, they return a Simulink.NumericType object. If you have existing models that use these functions as parameters to dialog boxes, the models continue to run as before and there is no need to change any model settings.

These functions do not offer full Data Type Assistant support. To benefit from this support, use fixdt instead.

Function	Return Value in Previous Releases — MATLAB structure	Return Value Effective R2013b — `NumericType`
`float('double')`	`Class: 'DOUBLE'`	`DataTypeMode: 'Double'`
`float('single')`	`Class: 'SINGLE'`	`DataTypeMode: 'Single'`

`sfix(16)`	`Class: 'FIX'`
	`IsSigned: 1`
	`MantBits: 16`
		`DataTypeMode: 'Fixed-point: unspecified scaling'`
		`Signedness: 'Signed'`
		`WordLength: 16`

`ufix(7)`	`Class: 'FIX'`
	`IsSigned: 0`
	`MantBits: 7`
		`DataTypeMode: 'Fixed-point: unspecified scaling'`
		`Signedness: 'Unsigned'`
		`WordLength: 7`

Function	Return Value in Previous Releases — MATLAB structure	Return Value Effective R2013b — `NumericType`

`sfrac(33,5)`	`Class: 'FRAC'`
	`IsSigned: 1`
	`MantBits: 33`
	`GuardBits: 5`
		`DataTypeMode: 'Fixed-point: binary point scaling'`
		`Signedness: 'Signed'`
		`WordLength: 33`
		`FractionLength: 27`

`ufrac(44)`	`Class: 'FRAC'`
	`IsSigned: 0`
	`MantBits: 44`
	`GuardBits: 0`
		`DataTypeMode: 'Fixed-point: binary point scaling'`
		`Signedness: 'Unsigned'`
		`WordLength: 44`
		`FractionLength: 44`

`sint(55)`	`Class: 'INT'`
	`IsSigned: 1`
	`MantBits: 55`
		`DataTypeMode: 'Fixed-point: binary point scaling'`
		`Signedness: 'Signed'`
		`WordLength: 55`
		`FractionLength: 0`

`uint(77)`	`Class: 'INT'`
	`IsSigned: 0`
	`MantBits: 77`
		`DataTypeMode: 'Fixed-point: binary point scaling'`
		`Signedness: 'Unsigned'`
		`WordLength: 77`
		`FractionLength: 0`

Compatibility Considerations

MATLAB Code

MATLAB code that depends on the return arguments of these functions being a structure with fields named `Class`, `MantBits` or `GuardBits` no longer works correctly. Change the code to access the appropriate properties of a `NumericType` object, for example, `DataTypeMode`, `Signedness`, `WordLength`, `FractionLength`, `Slope` and `Bias`.

C Code

Update C code that expects the data type of parameters to be a legacy structure to handle `NumericType` objects instead. For example, if you have S-functions that take legacy structures as parameters, update these S-functions to accept `NumericType` objects.

MAT-files

Effective R2013b, if you open a Simulink model that uses a MAT-file that contains a data type specification created using the legacy functions, the model uses the same data types and behaves in the same way as in previous releases but Simulink generates a warning.

To eliminate the warning, recreate the data type specifications using NumericType objects and save the MAT-file.

You can use the fixdtupdate function to update a data type specified using the legacy structure to use a NumericType. For example, if you saved a data type specification in a MAT-file as follows in a previous release:

```
oldDataType = sfrac(16);
save myDataTypeSpecification oldDataType
```

use fixdtUpdate to recreate the data type specification to use NumericType:

```
load DataTypeSpecification
fixdtUpdate(oldDataType)

ans =

  NumericType with properties:

      DataTypeMode: 'Fixed-point: binary point scaling'
        Signedness: 'Signed'
        WordLength: 16
    FractionLength: 15
           IsAlias: 0
         DataScope: 'Auto'
        HeaderFile: ''
       Description: ''
```

For more information, at the MATLAB command line, enter:

```
fixdtUpdate
```

numberofelements function being removed in a future release

The numberofelements function will be removed in a future release of Fixed-Point Designer software. Use numel instead.

R2013a

Version: 4.0

New Features

Bug Fixes

Compatibility Considerations

Product restructuring

The Fixed-Point Designer product replaces two pre-existing products: Fixed-Point Toolbox™ and Simulink Fixed Point™. You can access archived documentation for both products on the MathWorks® Web site.

Histogram logging in instrumented MATLAB Code Generation report

The `buildInstrumentedMex` and `showInstrumentationResults` instrumentation functions now can generate log2 histograms. A histogram is generated for each named and intermediate variable and for each expression in your code. The code generation report **Variables** tab includes a link to the histogram for each variable. You can use this histogram to determine the word and fraction lengths for your fixed-point values. Refer to the buildInstrumentedMex and showInstrumentationResults reference pages for information.

`fi` object in indexing and switch-case expressions

Effective this release, you can use `fi` objects as indices to arrays of built-in types and `fi` types. You can also use `fi` objects in switch-case expressions. These changes let you use `fi` objects without having to convert them. See the fi reference page for examples.

`zeros`, `ones`, and `cast` code reuse for floating-point and fixed-point types

The `zeros`, `ones`, and `cast` functions now work with fixed-point data types as well as built-in data types. The functions can now return an output whose class matches that of a specified numeric variable or `fi` object. For built-in data types, the output assumes the numeric data type, sparsity, and complexity (real or complex) of the specified numeric variable. For `fi` objects, the output assumes the `numerictype`, complexity (real or complex), and `fimath` of the specified `fi` object.

For example:

```
>> a = fi([],1,24,12);
>> c = cast(pi,'like',a)

c =
```

```
    3.1416

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 24
        FractionLength: 12

>> z = zeros(2,3,'like',a)

z =

     0     0     0
     0     0     0

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 24
        FractionLength: 12

>> o = ones(2,3,'like',a)

o =

     1     1     1
     1     1     1

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 24
        FractionLength: 12
```

This capability allows you to cleanly separate algorithm code in MATLAB from data type specifications. Using separate data type specifications enables you to:

- Reuse your algorithm code with different data types.
- Switch easily between fixed-point and floating-point data types to compare fixed-point behavior to a floating-point baseline.
- Try different fixed-point data types to determine their effect on the behavior of your algorithm.
- Write clean, readable code.

For more information, see Implement FIR Filter Algorithm for Floating-Point and Fixed-Point Types using cast and zeros.

Code generation for `x.^n` when n is a variable and `x` is a `fi` object

If the output type can be derived from the input settings, the `mpower` and `power` functions no longer require a constant exponent input. For more information, see mpower and power.

Fixed-Point Advisor support for model reference

The Fixed-Point Advisor now performs checks on referenced models. It checks the entire model reference hierarchy against fixed-point guidelines. The Advisor also provides guidance about model configuration settings and unsupported blocks to help you prepare your model for conversion to fixed point.

Automated conversion of floating-point to fixed-point types in MATLAB Coder projects

You can now convert floating-point MATLAB code to fixed-point C code using the fixed-point conversion capability in MATLAB Coder projects. You can choose to propose data types based on simulation range data, static range data, or both.

Note: You must have a MATLAB Coder license.

During fixed-point conversion, you can:

- Propose fraction lengths based on default word lengths.
- Propose word lengths based on default fraction lengths.
- Optimize whole numbers.
- Specify safety margins for simulation min/max data.
- Validate that you can build your project with the proposed data types.
- Test numerics by running the test file with the fixed-point types applied.
- View a histogram of bits used by each variable.

For more information, see Propose Fixed-Point Data Types Based on Simulation Ranges and Propose Fixed-Point Data Types Based on Derived Ranges.

Improved autoscaling for models with virtual bus signals

Autoscaling with the Fixed-Point Tool now handles data type constraints for virtual buses that do not have any associated bus objects. The data type proposals take into account the constraints introduced by these bus signals.

This improved autoscaling reduces data type mismatch errors. It also enables the Fixed-Point Tool to provide additional diagnostic information when you accept autoscaling proposals. For more information, see Shared Data Type Summary.

Data Type Override for MATLAB Function block using built-in doubles and singles

The data type override rules for MATLAB Function block input signals and parameters have changed. If the input signals and parameters are `double` or `single`, and you specify data type override to be `Double` or `Single`, the overridden data types are now built-in `double` or built-in `single`, not `fi double` and `fi single` as in previous releases. If the input signals and parameters are `fi` objects or fixed-point signals, and you specify data type override to be `Double` or `Single`, the overridden data types are `fi double` and `fi single` as in previous releases. For more information, see MATLAB Function Block with Data Type Override.

Compatibility Considerations

If you have MATLAB Function block code from previous releases that contains special cases for `fi double` or `fi single`, and you specify data type override to be `Double` or `Single`, you might have to update this code to handle built-in `double` and `single`.

Instrumentation for arrays of structs

The buildInstrumentedMex and showInstrumentationResults instrumentation functions now show instrumentation results for arrays of structs. Each field of each struct is logged and appears in the code generation report on the **Variables** tab.

File I/O function support

The following file I/O functions are now supported for code acceleration and generation:

- fclose

- fopen
- fprintf

To view implementation details, see Functions Supported for Code Acceleration or Generation.

Support for nonpersistent handle objects

You can now accelerate code using `fiaccel` for local variables that contain references to handle objects or System objects. In previous releases, accelerating code for these objects was limited to objects assigned to persistent variables.

Load from MAT-files for code acceleration

`fiaccel` now supports a subset of the load function for loading run-time values from a MAT-file. It also provides a new function, coder.load, for loading compile-time constants. This support facilitates code generation from MATLAB code that uses `load` to load constants into a function. You no longer have to manually type in constants that were stored in a MAT-file.

To view implementation details for the `load` function, see Functions Supported for Code Acceleration or Generation.

New toolbox functions supported for code acceleration and generation

To view implementation details, see Functions Supported for Code Acceleration or Generation.

Bitwise Operation Functions

- flintmax

Computer Vision System Toolbox Classes and Functions

- binaryFeatures
- insertMarker
- insertShape

Data File and Management Functions

- computer

- fclose
- fopen
- fprintf
- load

Image Processing Toolbox Functions

- conndef
- imcomplement
- imfill
- imhmax
- imhmin
- imreconstruct
- imregionalmax
- imregionalmin
- iptcheckconn
- padarray

Interpolation and Computational Geometry

- interp2

MATLAB Desktop Environment Functions

- ismac
- ispc
- isunix

String Functions

- strfind
- strrep

Function to be removed in a future release

The `saveglobalfimathpref` will be removed in a future release.

Compatibility Considerations

Do not save `globalfimath` as a MATLAB preference. If you have previously saved `globalfimath` as a MATLAB preference, use `removeglobalfimathpref` to remove it.

Function being removed

The `emlmex` function has been removed.

Compatibility Considerations

The `emlmex` function generates an error in R2013a. Use fiaccel instead.