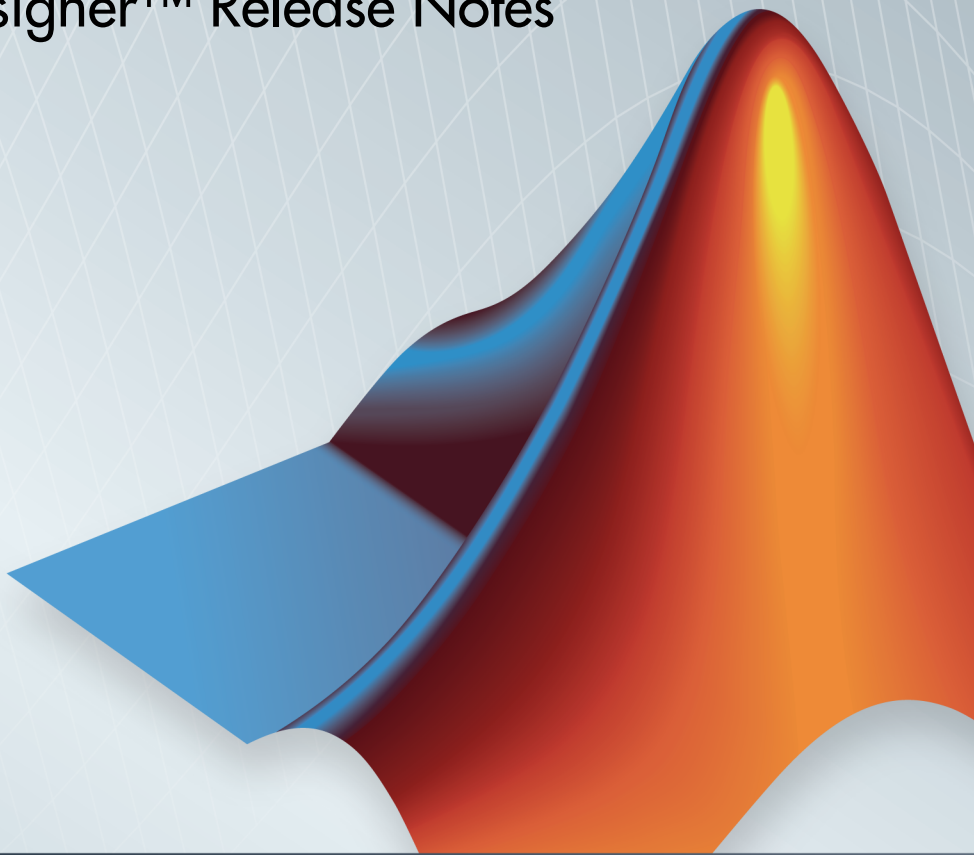
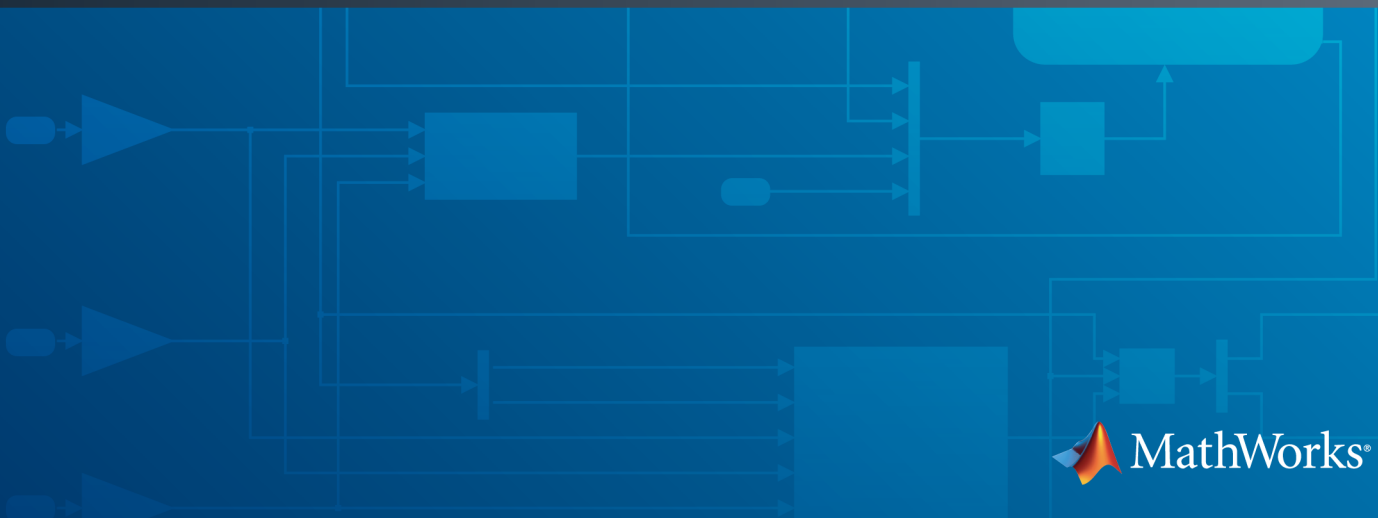


Fixed-Point Designer™ Release Notes



MATLAB®



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 2013–2014 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Fixed-Point Converter app for automated conversion of floating-point MATLAB code	1-2
Commands for scripting fixed-point conversion and accessing the collected data in Simulink	1-2
Automated fixed-point conversion for commonly used DSP System objects, including Biquad Filter, FIR Filter, and FIR Rate Converter	1-3
Simulation range collection and data type proposals for MATLAB Function blocks in Simulink	1-3
Overflow diagnostics to distinguish between wrap and saturation in Simulink	1-3
Highlighting of potential data type issues in generated HTML report	1-4
Code generation of for loops using fixed-point loop indices .	1-4
Cast net slope computations using rational numbers	1-4
Lock Column View option in the Fixed-Point Tool	1-4
Fixed-Point Advisor enhancements	1-5
hdlram renamed hdl.RAM	1-5
Changes to data type strings	1-5
Signal data type display	1-5
tostring function now uses 0 and 1 to represent signedness . .	1-5

New featured examples	1-6
------------------------------------	------------

R2014a

Data type override and automatic data typing for bus objects	2-2
Data type override for bus objects	2-2
Autoscaling for bus objects	2-2
Derived ranges for complex signals in Simulink	2-2
cordicsqrt function for fixed-point CORDIC-based square root functionality	2-2
Overflow detection with scaled double data types in MATLAB Coder projects	2-3
Fixed-point ARM Cortex-M code replacement support for DSP System Toolbox FIR filters	2-3
Fixed-Point Advisor support for referenced configuration sets	2-3
Enhancements to automated conversion of MATLAB code .	2-3
Support for MATLAB classes	2-3
Generated fixed-point code enhancements	2-4
Fixed-point report	2-4
Automatic C compiler setup	2-4
More flexible control of dsp.LMSFilter System object fixed-point settings	2-4
Derived ranges for For Each and For Each Subsystem blocks	2-5

C99 long long integer data type for embedded code generation	3-2
Model Advisor fixed-point checks with additional coverage and optimization awareness	3-2
fi object as an index in colon expressions and an argument to numel and bit index functions	3-3
fi object as an index in colon expressions	3-3
fi objects as bit index input argument	3-3
fi objects as shift-value input argument	3-3
numel function support for fi inputs	3-3
Improved efficiency of data type internal rules for Lookup Table blocks	3-3
Derived ranges for complex variables in MATLAB Coder projects	3-4
Simplified modeling of single-precision designs	3-4
Range analysis support on Mac platforms	3-5
Changes to showInstrumentationResults function options .	3-5
New option to suppress display of MATLAB code	3-5
Removal of -browser option	3-5
Changes to Continuous state-space block family range analysis support	3-5
Enhanced fiaccel support for int64 and uint64 functions ...	3-6
Support for LCC compiler on Microsoft Windows (64-bit) machines	3-6
Warning for use of inexact fi and fimath property names .	3-6
Conversion of numeric variables into Simulink.Parameter objects	3-6

Fixed-point conversion test file coverage results	3-7
Fixed-point conversion workflow supports designs that use enumerated types	3-7
Fixed-point conversion of variable-size data using simulation ranges	3-7
Error checking improvements for bitconcat, bitandreduce, bitorreduce, bitxorreduce, bitsliceget functions	3-7
Legacy data type specification functions return numeric objects	3-8
numberofelements function being removed in a future release	3-10

R2013a

Product restructuring	4-2
Histogram logging in instrumented MATLAB Code Generation report	4-2
fi object in indexing and switch-case expressions	4-2
zeros, ones, and cast code reuse for floating-point and fixed-point types	4-2
Code generation for $x.^n$ when n is a variable and x is a fi object	4-4
Fixed-Point Advisor support for model reference	4-4
Automated conversion of floating-point to fixed-point types in MATLAB Coder projects	4-4
Improved autoscaling for models with virtual bus signals ..	4-5

Data Type Override for MATLAB Function block using built-in doubles and singles	4-5
Instrumentation for arrays of structs	4-5
File I/O function support	4-5
Support for nonpersistent handle objects	4-6
Load from MAT-files for code acceleration	4-6
New toolbox functions supported for code acceleration and generation	4-6
Function to be removed in a future release	4-8
Function being removed	4-8

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: 1.99999999254942 — Stored integer value is $(2^{128} - 2^{100})$.
- Output type: `single`

Release	Calculation performed by Fixed-Point Designer	Output Result	Design Goal
R2013b	<code>Y = single(2⁻¹²⁷) * single(2¹²⁸-2¹⁰⁰)</code> <code>= single(2⁻¹²⁷) * Inf</code>	Inf	Strict singles
Previous releases	<code>Y = single(double(2⁻¹²⁷) * double(2¹²⁸ - 2¹⁰⁰))</code> <code>= single(2⁻¹²⁷ * 3.402823656532e+38)</code>	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 — <code>NumericType</code>
<code>float('double')</code>	Class: 'DOUBLE'	DataTypeMode: 'Double'
<code>float('single')</code>	Class: 'SINGLE'	DataTypeMode: 'Single'
<code>sfix(16)</code>	Class: 'FIX' IsSigned: 1 MantBits: 16	DataTypeMode: 'Fixed-point: unspecified scaling' Signedness: 'Signed' WordLength: 16
<code>ufix(7)</code>	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

- `strep`

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.