Release Notes for Fixed-Point Designer™

How to Contact MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Release Notes for Fixed-Point Designer™

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

R2013a

R2013a

Version: 4.0
New Features: Yes
Bug Fixes: Yes

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

Compatibility Considerations: Yes

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`.

MATLAB Function block uses `DataTypeOverrideAppliesTo` setting

When you specify data type override for subsystems that contain a MATLAB Function block, you can now also specify which data types to override in the block. Use the `DataTypeOverrideAppliesTo` parameter to specify whether to override all numeric types, floating-point data types only, or fixed-point data types only. In previous releases, the `DataTypeOverrideAppliesTo` parameter had no effect on MATLAB Function block input and parameter arguments.

You can specify data type override settings at the model or subsystem level. For more information, see "MATLAB Function Block with Data Type Override".

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

Compatibility Considerations: Yes

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
Compatibility Considerations: Yes

The `emlmex` function has been removed.

Compatibility Considerations

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