# MATLAB® Coder™ Release Notes

### How to Contact MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

*MATLAB® Coder™ Release Notes*

© COPYRIGHT 2011–2013 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

## R2013b



## R2013a

## R2012b



## R2012a

## R2011b



## R2011a

# R2013b

**Version: 2.5**

**New Features: Yes**

**Bug Fixes: Yes**

## Code generation for Statistics Toolbox and Phased Array System Toolbox

Code generation now supports more than 100 Statistics Toolbox™ functions. For implementation details, see “Statistics Toolbox Functions”.

Code generation now supports most of the Phased Array System Toolbox™ functions and System objects. For implementation details, see “Phased Array System Toolbox Functions” and “Phased Array System Toolbox System Objects”.

## Toolbox functions supported for code generation

For implementation details, see “Functions Supported for C/C++ Code Generation — Alphabetical List”.

### Data Type Functions

- `narginchk`

### Programming Utilities

- `mfilename`

### Specialized Math

- `psi`

### Computer Vision System Toolbox Classes and Functions

- `extractFeatures`
- `detectSURFFeatures`
- `disparity`
- `detectMSERFeatures`
- `detectFASTFeatures`
- `vision.CascadeObjectDetector`

- `vision.PointTracker`
- `vision.PeopleDetector`
- `cornerPoints`
- `MSERRegions`
- `SURFPoints`

## `parfor` function for standalone code generation, enabling execution on multiple cores

You can use MATLAB® Coder™ software to generate standalone C/C++ code from MATLAB code that contains `parfor`-loops. The code generation software uses the Open Multi-Processing (OpenMP) application interface to generate C/C++ code that runs in parallel on multiple cores on the target hardware.

For more information, see `parfor` and "Accelerate MATLAB Algorithms That Use Parallel for-loops (parfor)".

## Persistent variables in `parfor`-loops

You can now generate code from parallel algorithms that use persistent variables.

For more information, see `parfor`.

## Random number generator functions `in` `parfor`-loops

You can now generate code from parallel algorithms that use the random number generators `rand`, `randn`, `randi`, `randperm`, and `rng`.

For more information, see `parfor`.

## External code integration using `coder.ExternalDependency`

You can define the interface to external code using the new `coder.ExternalDependency` class. Methods of this class update the compile and build information required to integrate the external code with MATLAB code. In your MATLAB code, you can call the external code without needing to update build information. See `coder.ExternalDependency`.

## Updating build information using `coder.updateBuildInfo`

You can use the new function `coder.updateBuildInfo` to update build information. For example:

```
coder.updateBuildInfo('addLinkFlags','/STACK:1000000');
```

adds a stack size option to the linker command line. See `coder.updateBuildInfo`.

## Generation of simplified code using built-in C types

**Compatibility Considerations: Yes**

By default, MATLAB Coder now uses built-in C types in the generated code. You have the option to use predefined types from `rtwtypes.h`. To control the data type in the generated code:

- In a project, on the Project Settings dialog box **Code Appearance** tab, use the **Data Type Replacement** setting.
- At the command line, use the configuration object parameter `DataTypeReplacement`.

The built-in C type that the code generation software uses depends on the target hardware.

For more information, see “Specify Data Type Used in Generated Code”.

### Compatibility Considerations

If you use the default configuration or project settings, the generated code has built-in C types such as `double` or `char`. Code generated prior to R2013b has predefined types from `rtwtypes.h`, such as `real_T` or `int32_T`.

## Conversion of MATLAB expressions into C constants using `coder.const`

You can use the new function `coder.const` to convert expressions and function calls to constants at compile time. See `coder.const` and "Constant Folding".

## Highlighting of constant function arguments in the compilation report

The compilation report now highlights constant function arguments and displays them in a distinct color. You can display the constant argument data type and value by placing the cursor over the highlighted argument. You can export the constant argument value to the base workspace where you can display detailed information about the argument.

For more information, see "Viewing Variables in Your MATLAB Code".

## Code Generation Support for int64, uint64 data types

You can now use `int64` and `uint64` data types for code generation.

## C99 long long integer data type for code generation

If your target hardware and 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 and multiword helper functions. To specify the long long data type for code generation:

- In a project, on the Project Settings dialog box **Hardware** tab, use the following production and test hardware settings:
  - **Enable long long**: Specify that your C compiler supports the long long data type. Set to `Yes` to enable **Sizes: long long**.
  - **Sizes: long long**: Describe length in bits of the C long long data type supported by the hardware.

- At the command line, use the following hardware implementation configuration object parameters:
  - `ProdLongLongMode`: Specify that your C compiler supports the long long data type. Set to `true` to enable `ProdBitPerLongLong`.
  - `ProdBitPerLongLong`: Describes the length in bits of the C long long data type supported by the production hardware.
  - `TargetLongLongMode`: Specifies whether your C compiler supports the long long data type. Set to `true` to enable `TargetBitPerLongLong`.
  - `TargetBitPerLongLong`: Describes the length in bits of the C long long data type supported by the test hardware.

  For more information, see the class reference information for `coder.HardwareImplementation`.

## Change to passing structures by reference

**Compatibility Considerations: Yes**

In R2013b, the option to pass structures by reference to entry-point functions in the generated code applies to function outputs and function inputs. In R2013a, this option applied only to inputs to entry-point functions.

### Compatibility Considerations

If you select the pass structures by reference option, and a MATLAB entry-point function has a single output that is a structure, the generated C function signature in R2013b differs from the signature in R2013a. In R2013a, the generated C function returns the output structure. In R2013b, the output structure is a pass by reference function parameter.

If you have code that calls one of these functions generated in R2013a, and then you generate the function in R2013b, you must change the call to the function. For example, suppose `S` is a structure in the following MATLAB function `foo`.

```
function S = foo()
```

If you generate this function in R2013a, you call the function this way:

```
S = foo();
```

If you generate this function in R2013b, you call the function this way:

```
foo(&S);
```

### `coder.runTest` new syntax

Use the syntax `coder.runTest(test_fcn, MEX_name_ext)` to run `test_fcn` replacing calls to entry-point functions with calls to the corresponding MEX functions in the MEX file named `MEX_name_ext`. `MEX_name_ext` includes the platform-specific file extension. See `coder.runTest`.

### `coder.target` syntax change

The new syntax for `coder.target` is:

```
tf = coder.target('target')
```

For example, `coder.target('MATLAB')` returns true when code is running in MATLAB. See `coder.target`.

You can use the old syntax, but consider changing to the new syntax. The old syntax will be removed in a future release.

## Changes for complex values with imaginary part equal to zero

**Compatibility Considerations: Yes**

In R2013b, complex values with an imaginary part equal to zero become real when:

- They are returned by a MEX function.
- They are passed to an extrinsic function.

See “Expressions With Complex Operands Yield Complex Results”.

### Compatibility Considerations

MEX functions generated in R2013b return a real value when a complex result has an imaginary part equal to zero. MEX functions generated prior to R2013b return a complex value when a complex result has an imaginary part equal to zero.

In R2013b, complex values with imaginary part equal to zero become real when passed to an extrinsic function. In previous releases, they remain complex.

## Support for LCC compiler on Windows 64-bit machines

The LCC-win64 compiler is shipping with MATLAB Coder for Microsoft® Windows® 64-bit machines. For Windows 64-bit machines that do not have a third-party compiler installed, MEX code generation uses LCC by default.

You cannot use LCC for code generation of C/C++ static libraries, C/C++ dynamic libraries, or C/C++ executables. For these output types, you must install a compiler. See `http://www.mathworks.com/support/compilers/current_release/`.

## Fixed-Point conversion enhancements

These capabilities require a Fixed-Point Designer™ license.

### Fixed-Point conversion option for `codegen`

You can now convert floating-point MATLAB code to fixed-point code, and then generate C/C++ code at the command line using the option

`-float2fixed` with the `codegen` command. See `codegen` and "Convert Floating-Point MATLAB Code to Fixed-Point C Code Using codegen".

### Fixed-point conversion using derived ranges on Mac platforms

You can now convert floating-point MATLAB code to fixed-point C code using the Fixed-Point Conversion tool in MATLAB Coder projects on Mac platforms.

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

### Derived ranges for complex variables in MATLAB Coder projects

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

### Fixed-point conversion workflow supports designs that use enumerated types

Using the Fixed-Point Conversion tool in MATLAB Coder projects, you can 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".

### 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".

### Fixed-point conversion test file coverage results

The 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.

For more information, see “Code Coverage”.

# Check bug reports for issues and fixes

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at `www.mathworks.com/support/bugreports/`. Use the Saved Searches and Watched Bugs tool with the search phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

## Search R2013b Bug Reports

Known Bugs for Incorrect Code Generation:
`www.mathworks.com/support/bugreports/?product=ALL&release=R2013b&keyword=Incorrect+Code+Generation`

All Known Bugs for This Product:
`www.mathworks.com/support/bugreports/?release=R2013b&product=ME`

# R2013a

**Version: 2.4**

**New Features: Yes**

**Bug Fixes: Yes**

## Automatic fixed-point conversion during code generation (with Fixed-Point Designer)

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 Fixed-Point Designer 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.

## File I/O function support

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

- `fclose`
- `fopen`
- `fprintf`

To view implementation details, see Functions Supported for Code Generation — Alphabetical List.

## Support for nonpersistent handle objects

You can now generate code for local variables that contain references to handle objects or System objects. In previous releases, generating code for these objects was limited to objects assigned to persistent variables.

## Structures passed by reference to entry-point functions

You can now specify to pass structures by reference to entry-point functions in the generated code. This optimization is available for standalone code generation only; it is not available for MEX functions. Passing structures by reference reduces the number of copies at entry-point function boundaries in your generated code. It does not affect how structures are passed to functions other than entry-point functions.

To pass structures by reference:

- In a project, on the Project Settings dialog box **All Settings** tab, under **Advanced**, set **Pass structures by reference to entry-point functions** to `Yes`.
- At the command line, create a code generation configuration object and set the `PassStructByReference` parameter to `true`. For example:

```
cfg = coder.config('lib');
cfg.PassStructByReference=true;
```

## Include custom C header files from MATLAB code

The `coder.cinclude` function allows you to specify in your MATLAB code which custom C header files to include in the generated C code. Each header file that you specify using `coder.cinclude` is included in every C/C++ file generated from your MATLAB code. You can specify whether the `#include` statement uses double quotes for application header files or angle brackets for system header files in the generated code.

For example, the following code for function `foo` specifies to include the application header file `mystruct.h` in the generated code using double quotes.

```
function y = foo(x1, x2)
%#codegen
coder.cinclude('mystruct.h');

...
```

For more information, see `coder.cinclude`.

## Load from MAT-files

MATLAB Coder now supports a subset of the `load` function for loading run-time values from a MAT-file while running a MEX function. It also provides a new function, `coder.load`, for loading compile-time constants when generating MEX or standalone code. 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 Generation — Alphabetical List.

For more information, see `coder.load`.

## `coder.opaque` function enhancements

When you use `coder.opaque` to declare a variable in the generated C code, you can now also specify the header file that defines the type of the variable. Specifying the location of the header file helps to avoid compilation errors because the MATLAB Coder software can find the type definition more easily.

You can now compare `coder.opaque` variables of the same type. This capability helps you verify, for example, whether an `fopen` command succeeded.

```
null = coder.opaque('FILE*','NULL','HeaderFile','stdio.h');
ftmp = null;
ftmp = coder.ceval('fopen',fname,permission);
```

```
if ftmp == null
  % Error - file open failed
end
```

For more information, see `coder.opaque`.

## Automatic regeneration of MEX functions in projects

When you run a test file from a MATLAB Coder project to verify the behavior of the generated MEX function, the project now detects when to rebuild the MEX function. MATLAB Coder rebuilds the MEX function only if you have modified the original MATLAB algorithm since the previous build, saving you time during the verification phase.

## MEX function signatures include constant inputs

**Compatibility Considerations: Yes**

When you generate a MEX function for a MATLAB function that takes constant inputs, by default, the MEX function signature now contains the constant inputs. If you are verifying your MEX function in a project, this behavior allows you to use the same test file to run the original MATLAB algorithm and the MEX function.

### Compatibility Considerations

In previous releases, MATLAB Coder removed the constants from the MEX function signature. To use these existing scripts with MEX functions generated using R2013a software, do one of the following:

- Update the scripts so that they no longer remove the constants.
- Configure MATLAB Coder to remove the constant values from the MEX function signature.

To configure MATLAB Coder to remove the constant values:

- In a project, on the Project Settings dialog box **All Settings** tab, under **Advanced**, set **Constant Inputs** to `Remove from MEX signature`.
- At the command line, create a code generation configuration object, and, set the `ConstantInputs` parameter to `'Remove'`. For example:

```
cfg = coder.config;
cfg.ConstantInputs='Remove';
```

## Custom toolchain registration

**Compatibility Considerations: Yes**

MATLAB Coder software enables you to register third-party software build tools for creating executables and libraries.

- The software automatically detects supported tool chains on your system.
- You can manage and customize multiple tool chain definitions.
- Before generating code, you can select any one of the definitions using a drop-down list.
- The software generates simplified makefiles for improved readability.

For more information:

- See Custom Toolchain Registration.
- See the Adding a Custom Toolchain example.

### Compatibility Considerations

If you open a MATLAB Coder project or use a code generation configuration object from R2012b, the current version of MATLAB Coder software automatically tries to use the toolchain approach. If an existing project or configuration object does not use default target makefile settings, MATLAB Coder might not be able to upgrade to use a toolchain approach and will emit a warning. For more information, see Project or Configuration is Using the Template Makefile.

## Complex trigonometric functions

Code generation support has been added for complex `acosD`, `acotD`, `acscD`, `asecD`, `asinD`, `atanD`, `cosD`, `cscD`, `cotD`, `secD`, `sinD`, and `tanD` functions.

## `parfor` function reduction improvements and C support

When generating MEX functions for `parfor`-loops, you can now use `intersect` and `union` as reduction functions, and the following reductions are now supported:

- Concatenations
- Arrays
- Function handles

By default, when MATLAB Coder generates a MEX function for MATLAB code that contains a `parfor`-loop, MATLAB Coder no longer requires C++ and now honors the target language setting.

## Support for integers in number theory functions

Code generation supports integer inputs for the following number theory functions:

- `cumprod`
- `cumsum`
- `factor`
- `factorial`
- `gcd`
- `isprime`
- `lcm`
- `median`

- `mode`
- `nchoosek`
- `nextpow2`
- `primes`
- `prod`

To view implementation details, see Functions Supported for Code Generation — Alphabetical List.

## Enhanced support for class property initial values

**Compatibility Considerations: Yes**

If you initialize a class property, you can now assign a different type to the property when you use the class. For example, class `foo` has a property `prop1` of type `double`.

```
classdef foo %#codegen
  properties
    prop1= 0;
  end
  methods
    ...
  end
end
```

Function `bar` assigns a different type to `prop1`.

```
fumction bar %#codegen
  f=foo;
  f.prop1=single(0);
  ...
```

In R2013a, MATLAB Coder ignores the initial property definition and uses the reassigned type. In previous releases, MATLAB Coder did not support this reassignment and code generation failed.

### Compatibility Considerations

In previous releases, if the reassigned property had the same type as the initial value but a different size, the property became variable-size in the generated code. In R2013a, MATLAB Coder uses the size of the reassigned property, and the size is fixed. If you have existing MATLAB code that relies on the property being variable-size, you cannot generate code for this code in R2013a. To fix this issue, do not initialize the property in the property definition block.

For example, you can no longer generate code for the following function `bar`.

Class `foo` has a property `prop1` which is a scalar `double`.

```
classdef foo %#codegen
  properties
    prop1= 0;
  end
  methods
    ...
  end
end
```

Function `bar` changes the size of `prop1`.

```
fumction bar %#codegen
  f=foo;
  f.prop1=[1 2 3];
  % Use f
  disp(f.prop1);
  f.prop1=[1 2 3 4 5 6 ];
```

## Optimized generated code for `x=[x c]` when `x` is a vector

MATLAB Coder now generates more optimized code for the expression `x=[x c]`, if:

- `x` is a row or column vector.
- `x` is not in `c`.

- `x` is not aliased.
- There are no function calls in `c`.

In previous releases, the generated code contained multiple copies of `x`. In R2013a, it does not contain multiple copies of `x`.

This enhancement reduces code size and execution time. It also improves code readability.

## Default use of Basic Linear Algebra Subprograms (BLAS) libraries

**Compatibility Considerations: Yes**

MATLAB Coder now uses BLAS libraries whenever they are available. There is no longer an option to turn off the use of these libraries.

### Compatibility Considerations

If existing configuration settings disable BLAS, MATLAB Coder now ignores these settings.

## Changes to compiler support

**Compatibility Considerations: Yes**

MATLAB Coder supports these new compilers.

- On Microsoft Windows platforms, Visual C++® 11.
- On Mac OS X platforms, Apple Xcode 4.2 with Clang.

MATLAB Coder no longer supports the `gcc` compiler on Mac OS X platforms.

MATLAB Coder no longer supports Watcom for standalone code generation. Watcom is still supported for building MEX functions.

### Compatibility Considerations

- Because Clang is the only compiler supported on Mac OS X platforms, and Clang does not support Open MP, `parfor` is no longer supported on Mac OS X platforms.
- MATLAB Coder no longer supports Watcom for standalone code generation. Use Watcom only for building MEX functions. Use an alternative compiler for standalone code generation. For a list of supported compilers, see `http://www.mathworks.com/support/compilers/current_release/`.

## New toolbox functions supported for code generation

To view implementation details, see Functions Supported for Code Generation — Alphabetical List.

### 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`

## Functions being removed
**Compatibility Considerations: Yes**

These functions have been removed from MATLAB Coder software.

| Function Name | What Happens When You Use This Function? |
|---|---|
| `emlc` | Errors in R2013a. |
| `emlmex` | Errors in R2013a. |

## Compatibility Considerations

`emlc` and `emlmex` have been removed. Use `codegen` instead. If you have existing code that calls `emlc` or `emlmex`, use `coder.upgrade` to help convert your code to the new syntax.

## Check bug reports for issues and fixes

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at `www.mathworks.com/support/bugreports/`. Use the Saved Searches and Watched Bugs tool with the search phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

### Search R2013a Bug Reports

Known Bugs for Incorrect Code Generation:
`www.mathworks.com/support/bugreports/?product=ALL&release=R2013a&keyword=Incorrect+Code+Generation`

All Known Bugs for This Product:
`www.mathworks.com/support/bugreports/?release=R2013a&product=ME`

# R2012b

**Version: 2.3**

**New Features: Yes**

**Bug Fixes: Yes**

## `parfor` function support for MEX code generation, enabling execution on multiple cores

You can use MATLAB Coder software to generate MEX functions from MATLAB code that contains `parfor`-loops. The generated MEX functions can run on multiple cores on a desktop. For more information, see `parfor` and Acceleration of MATLAB Algorithms Using Parallel for-loops (parfor).

## Code generation readiness tool

The code generation readiness tool screens MATLAB code for features and functions that are not supported for code generation. The tool provides a report that lists the source files that contain unsupported features and functions and an indication of how much work is needed to make the MATLAB code suitable for code generation.

For more information, see `coder.screener` and Code Generation Readiness Tool.

## Reduced data copies and lightweight run-time checks for generated MEX functions

MATLAB Coder now eliminates data copies for built-in, non-complex data types. It also performs faster bounds checks. These enhancements result in faster generated MEX functions.

## Additional string function support for code generation

The following string functions are now supported for code generation. To view implementation details, see Functions Supported for Code Generation — Alphabetical List.

- `deblank`
- `hex2num`
- `isletter`

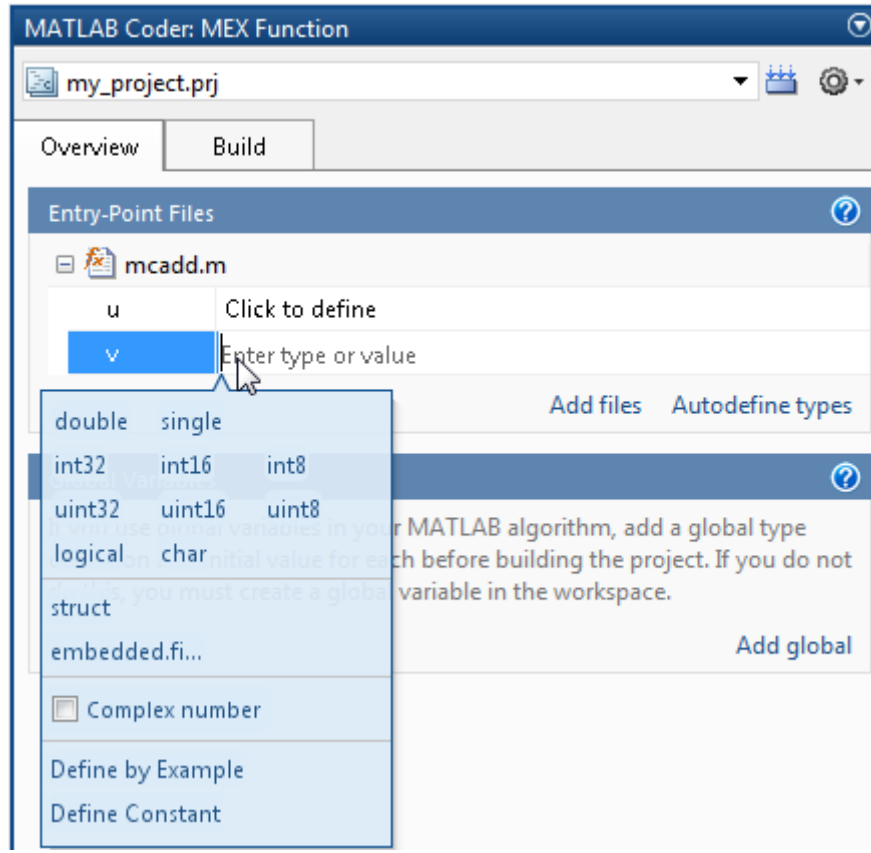
- `isspace`
- `isstrprop`
- `lower`
- `num2hex`
- `strcmpi`
- `strjust`
- `strncmp`
- `strncmpi`
- `strtok`
- `strtrim`
- `upper`

## Visualization functions in generated MEX functions

The MATLAB Coder software now detects calls to many common visualization functions, such as `plot`, `disp`, and `figure`. For MEX code generation, MATLAB Coder automatically calls out to MATLAB for these functions. For standalone code generation, MATLAB Coder does not generate code for these visualization functions. This capability reduces the amount of time that you spend making your code suitable for code generation. It also removes the requirement to declare these functions extrinsic using the `coder.extrinsic` function.

## Input parameter type specification enhancements

The updated project user interface facilitates input parameter type specification.




## Project import and export capability

You can now export project settings to a configuration object stored as a variable in the base workspace. You can then use the configuration object to import the settings into a different project or to generate code at the command line with the `codegen` function. This capability allows you to:

- Share settings between the project and command-line workflow
- Share settings between multiple projects
- Standardize on settings for code generation projects

For more information, see Share Build Configuration Settings.

## Package generated code in zip file for relocation

The `packNGo` function packages generated code files into a compressed zip file so that you can relocate, unpack, and rebuild them in another development environment. This capability is useful if you want to relocate files so that you can recompile them for a specific target environment or rebuild them in a development environment in which MATLAB is not installed.

For more information, see Package Code For Use in Another Development Environment.

## Fixed-point instrumentation and data type proposals

MATLAB Coder projects provide the following fixed-point conversion support:

- Option to generate instrumented MEX functions
- Use of instrumented MEX functions to provide simulation minimum and maximum results
- Fixed-point data type proposals based on simulation minimum and maximum values
- Option to propose fraction lengths or word lengths

You can use these proposed fixed-point data types to create a fixed-point version of your original MATLAB entry-point function.

---

**Note** Requires a Fixed-Point Toolbox™ license.

---

For more information, see Fixed-Point Conversion.

# New toolbox functions supported for code generation

To view implementation details, see Functions Supported for Code Generation — Alphabetical List.

### Computer Vision System Toolbox

- `integralImage`

### Image Processing Toolbox

- `bwlookup`
- `bwmorph`

### Interpolation and Computational Geometry

- `interp2`

### String Functions

- `deblank`
- `hex2num`
- `isletter`
- `isspace`
- `isstrprop`
- `lower`
- `num2hex`
- `strcmpi`
- `strjust`
- `strncmp`
- `strncmpi`
- `strtok`

- `strtrim`
- `upper`

### Trigonometric Functions

- `atan2d`

## New System objects supported for code generation

The following System objects are now supported for code generation. To see the list of System objects supported for code generation, see System Objects Supported for Code Generation.

### Communications System Toolbox

- `comm.ACPR`
- `comm.BCHDecoder`
- `comm.CCDF`
- `comm.CPMCarrierPhaseSynchronizer`
- `comm.GoldSequence`
- `comm.LDPCDecoder`
- `comm.LDPCEncoder`
- `comm.LTEMIMOChannel`
- `comm.MemorylessNonlinearity`
- `comm.MIMOChannel`
- `comm.PhaseNoise`
- `comm.PSKCarrierPhaseSynchronizer`
- `comm.RSDecoder`

### DSP System Toolbox

- `dsp.AllpoleFilter`
- `dsp.CICDecimator`

- dsp.CICInterpolator
- dsp.IIRFilter
- dsp.SignalSource

# Check bug reports for issues and fixes

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at `www.mathworks.com/support/bugreports/`. Use the Saved Searches and Watched Bugs tool with the search phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

## Search R2012b Bug Reports

Known Bugs for Incorrect Code Generation:
`www.mathworks.com/support/bugreports/?product=ALL&release=R2012b&keyword=Incorrect+Code+Generation`

All Known Bugs for This Product:
`www.mathworks.com/support/bugreports/?release=R2012b&product=ME`

# R2012a

**Version: 2.2**

**New Features: Yes**

**Bug Fixes: No**

## Code Generation for MATLAB Classes

In R2012a, there is preliminary support for code generation for MATLAB classes targeted at supporting System objects defined by users. For more information about generating code for MATLAB classes, see Code Generation for MATLAB Classes. For more information about generating code for System objects, see the DSP System Toolbox™, Computer Vision System Toolbox™ or the Communications System Toolbox™ documentation.

## Dynamic Memory Allocation Based on Size

**Compatibility Considerations: Yes**

By default, dynamic memory allocation is now enabled for variable-size arrays whose size exceeds a configurable threshold. This behavior allows for finer control over stack memory usage. Also, you can generate code automatically for more MATLAB algorithms without modifying the original MATLAB code.

### Compatibility Considerations

If you use scripts to generate code and you do not want to use dynamic memory allocation, you must disable it. For more information, see Controlling Dynamic Memory Allocation.

## C/C++ Dynamic Library Generation

You can now use MATLAB Coder to build a dynamically linked library (DLL) from the generated C code. These libraries are useful for integrating into existing software solutions that expect dynamically linked libraries.

For more information, see Generating C/C++ Dynamically Linked Libraries from MATLAB Code.

## Automatic Definition of Input Parameter Types

MATLAB Coder software can now automatically define input parameter types by inferring these types from test files that you supply. This capability

facilitates input type definition and reduces the risk of introducing errors when defining types manually.

To learn more about automatically defining types:

- In MATLAB Coder projects, see Autodefining Input Types.
- At the command line, see the `coder.getArgTypes` function reference pagehttp://www.mathworks.com/help/releases/R2012a/toolbox/coder/ref/coder.getargtyp

## Verification of MEX Functions

MATLAB Coder now provides support for test files to verify the operation of generated MEX functions. This capability enables you to verify that the MEX function is functionally equivalent to your original MATLAB code and to check for run-time errors.

To learn more about verifying MEX function behavior:

- In MATLAB Coder projects, see How to Verify MEX Functions in a Project.
- At the command line, see the `coder.runTest` function reference pagehttp://www.mathworks.com/help/releases/R2012a/toolbox/coder/ref/coder.runtest.h

## Enhanced Project Settings Dialog Box

The **Project Settings** dialog box now groups configuration parameters so that you can easily identify the parameters associated with code generation objectives such as speed, memory, and code appearance. The dialog boxes for code generation configuration objects, `coder.MexCodeConfig`, `coder.CodeConfig`, and `coder.EmbeddedCodeConfig`, also use the same new groupings.

To view the updated **Project Settings** dialog box:

1. In a project, click the **Build** tab.
2. On the **Build** tab, click the More settings link to open the **Project Settings** dialog box.

   For information about the parameters on each tab, click the **Help** button.

To view the updated dialog boxes for the code generation configuration objects:

1. At the MATLAB command line, create a configuration object. For example, create a configuration object for MEX code generation.

   ```
   mex_cfg = coder.config;
   ```

2. Open the dialog box for this object.

   ```
   open mex_cfg
   ```

   For information about the parameters on each tab, click the **Help** button.

## Projects Infer Input Types from assert Statements in Source Code

MATLAB Coder projects can now infer input data types from `assert` statements that define the properties of function inputs in your MATLAB entry-point files. For more information, see Defining Inputs Programmatically in the MATLAB File.

## Code Generation from MATLAB

For details about new toolbox functions and System objects supported for code generation, see the Code Generation from MATLAB Release Notes.

## New Demo

The following demo has been added:

| Demo... | Shows How You Can... |
|---|---|
| `coderdemo_reverb` | Generate a MEX function for an algorithm that uses MATLAB classes. |

# Check bug reports for issues and fixes

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at `www.mathworks.com/support/bugreports/`. Use the Saved Searches and Watched Bugs tool with the search phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

## Search R2012a Bug Reports

Known Bugs for Incorrect Code Generation:
`www.mathworks.com/support/bugreports/?product=ALL&release=R2012a&keyword=Incorrect+Code+Generation`

All Known Bugs for This Product:
`www.mathworks.com/support/bugreports/?release=R2012a&product=ME`

# R2011b

**Version: 2.1**

**New Features: Yes**

**Bug Fixes: No**

## Support for Deletion of Rows and Columns from Matrices

You can now generate C/C++ code from MATLAB code that deletes rows or columns from matrices. For example, the following code deletes the second column of matrix `X`:

```
X(:,2) = [];
```

For more information, see Diminishing the Size of a Matrix in the MATLAB documentation.

## Code Generation from MATLAB

For details of new toolbox functions and System objects supported for code generation, see Code Generation from MATLAB Release Notes.

# Check bug reports for issues and fixes

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at `www.mathworks.com/support/bugreports/`. Use the Saved Searches and Watched Bugs tool with the search phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

## Search R2011b Bug Reports

Known Bugs for Incorrect Code Generation:
`www.mathworks.com/support/bugreports/?product=ALL&release=R2011b&keyword=Incorrect+Code+Generation`

All Known Bugs for This Product:
`www.mathworks.com/support/bugreports/?release=R2011b&product=ME`

# R2011a

**Version: 2.0**

**New Features: Yes**

**Bug Fixes: No**

## New User Interface for Managing Projects

The new MATLAB Coder user interface simplifies the MATLAB to C/C++ code generation process. Using this user interface, you can:

- Specify the MATLAB files from which you want to generate code
- Specify the data types for the inputs to these MATLAB files
- Select an output type:
  - MEX function
  - C/C++ Static Library
  - C/C++ Executable
- Configure build settings to customize your environment for code generation
- Open the code generation report to view build status, generated code, and compile-time information for the variables and expressions in your MATLAB code

### To Get Started

You launch a MATLAB Coder project by doing one of the following:

- From the MATLAB main menu, select **File > New > Code Generation Project**
- Enter `coder` at the MATLAB command line

To learn more about working with MATLAB Coder, see Generating C Code from MATLAB Code Using the MATLAB Coder Project Interface.

## Migrating from Real-Time Workshop emlc Function

**Compatibility Considerations: Yes**

In MATLAB Coder, the `codegen` function replaces `emlc` with the following differences:

### New codegen Options

| Old emlc Option | New codegen Option |
|---|---|
| `-eg` | `-args` |
| `emlcoder.egc` | `coder.Constant` |
| `emlcoder.egs` | `coder.typeof(a,b,1)` specifies a variable-size input with the same class and complexity as `a` and same size and upper bounds as the size vector `b`.<br><br>Creates `coder.Type` objects for use with the `codegen -args` option. For more information, see `coder.typeof`. |
| `-F` | No `codegen` option available. Instead, use the default fimath. For more information, see the Fixed-Point Toolbox documentation. |
| `-global` | `-globals`<br><br>**Note** `-global` continues to work with `codegen` |
| `-N` | This option is no longer supported. Instead, set up `numerictype` in MATLAB. |
| `-s` | `-config`<br><br>Use with the new configuration objects, see "New Code Generation Configuration Objects" on page 50. |
| `-T rtw:exe` | `-config:exe`<br><br>Use this option to generate a C/C++ executable using default build options. Otherwise, use `-config` with a `coder.CodeConfig` or `coder.EmbeddedCodeConfig` configuration object. |

| Old emlc Option | New codegen Option |
|---|---|
| `-T mex` | `-config:mex`<br>Use this option to generate a MEX function using default build options. Otherwise, use `-config` with a `coder.MexCodeConfig` configuration object. |
| `-T rtw`<br>`-T rtw:lib` | `-config:lib`<br>Use either of these options to generate a C/C++ library using default build options. Otherwise, use `-config` with a `coder.CodeConfig` or `coder.EmbeddedCodeConfig` configuration object. |

### New Code Generation Configuration Objects

The `codegen` function uses new configuration objects that replace the old `emlc` objects with the following differences:

| Old emlc Configuration Object | New codegen Configuration Object |
|---|---|
| `emlcoder.MEXConfig` | `coder.MexCodeConfig` |
| `emlcoder.RTWConfig`<br>`emlcoder.RTWConfig('grt')` | `coder.CodeConfig`<br>The `SupportNonFinite` property is now available without an Embedded Coder® license.<br>The following property names have changed:<br>• `RTWCompilerOptimization` is now `CCompilerOptimization`<br>• `RTWCustomCompilerOptimization` is now `CCustomCompilerOptimization`<br>• `RTWVerbose` is now `Verbose` |

| Old emlc Configuration Object | New codegen Configuration Object |
|---|---|
| `emlcoder.RTWConfig('ert')` | `coder.EmbeddedCodeConfig`<br>The following property names have changed:<br>• `MultiInstanceERTCode` is now `MultiInstanceCode`<br>• `RTWCompilerOptimization` is now `CCompilerOptimization`<br>• `RTWCustomCompilerOptimization` is now `CCustomCompilerOptimization`<br>• `RTWVerbose` is now `Verbose` |
| `emlcoder.HardwareImplementation` | `coder.HardwareImplementation` |

## The codegen Function Has No Default Primary Function Input Type

In previous releases, if you used the `emlc` function to generate code for a MATLAB function with input parameters, and you did not specify the types of these inputs, by default, `emlc` assumed that these inputs were real, scalar, doubles. In R2011a, the `codegen` function does not assume a default type. You must specify at least the class of each primary function input. For more information, see Specifying Properties of Primary Function Inputs in a Project.

## Compatibility Considerations

If your existing script calls `emlc` to generate code for a MATLAB function that has inputs and does not specify the input types, and you migrate this script to use `codegen`, you must modify the script to specify inputs.

## The codegen Function Processes Compilation Options in a Different Order

In previous releases, the `emlc` function resolved compilation options from left to right so that the right-most option prevailed. In R2011a, the `codegen` function gives precedence to individual command-line options over options

specified using a configuration object. If command-line options conflict, the right-most option prevails.

### Compatibility Considerations

If your existing script calls `emlc` specifying a configuration object as well as other command-line options, and you migrate this script to use `codegen`, `codegen` might not use the same configuration parameter values as `emlc`.

## New coder.Type Classes

MATLAB Coder includes the following new classes to specify input parameter definitions:

- `coder.ArrayType`
- `coder.Constant`
- `coder.EnumType`
- `coder.FiType`
- `coder.PrimitiveType`
- `coder.StructType`
- `coder.Type`

## New coder Package Functions

The following new package functions let you work with objects and types for C/C++ code generation:

| Function | Purpose |
|---|---|
| `coder.config` | Create MATLAB Coder code generation configuration objects |
| `coder.newtype` | Create a new `coder.Type` object |

| Function | Purpose |
|---|---|
| `coder.resize` | Resize a `coder.Type` object |
| `coder.typeof` | Convert a MATLAB value into its canonical type |

## Script to Upgrade MATLAB Code to Use MATLAB Coder Syntax

The `coder.upgrade` script helps you upgrade to MATLAB Coder by searching your MATLAB code for old commands and options and replacing them with their new equivalents. For more information, at the MATLAB command prompt, enter `help coder.upgrade`.

## Embedded MATLAB Now Called Code Generation from MATLAB

MathWorks® is no longer using the term *Embedded MATLAB* to refer to the language subset that supports code generation from MATLAB algorithms. This nomenclature incorrectly implies that the generated code is used in embedded systems only. The new term is *code generation from MATLAB*. This terminology better reflects the full extent of the capability for translating MATLAB algorithms into readable, efficient, and compact MEX and C/C++ code for deployment to both desktop and embedded systems.

## MATLAB Coder Uses rtwTargetInfo.m to Register Target Function Libraries

In previous releases, the `emlc` function also recognized the customization file, `sl_customization.m`. In R2011a, the MATLAB Coder software does not recognize this customization file, you must use `rtwTargetInfo.m` to register a Target Function Library (TFL). To register a TFL, you must have Embedded Coder software. For more information, see Use the rtwTargetInfo API to Register a CRL with MATLAB Coder Software in the Embedded Coder documentation.

## New Getting Started Tutorial Video

To learn how to generate C code from MATLAB code, see the "Generating C Code from MATLAB Code" video in the MATLAB Coder Getting Started demos.

## New Demos

The following demos have been added:

| Demo... | Shows How You Can... |
|---|---|
| Hello World | Generate and run a MEX function from a simple MATLAB program |
| Working with Persistent Variables | Compute the average for a set of values by using persistent variables |
| Working with Structure Arrays | Shows how to build a scalar template before growing it into a structure array, a requirement for code generation from MATLAB. |
| Balls Simulation | Simulates bouncing balls and shows that you should specify only the entry function when you compile the application into a MEX function. |
| General Relativity with MATLAB Coder | Uses Einstein's theory of general relativity to calculate geodesics in curved space-time. |
| Averaging Filter | Generate a standalone C library from MATLAB code using `codegen` |
| Edge Detection on Images | Generate a standalone C library from MATLAB code that implements a Sobel filter |
| Read Text File | Generate a standalone C library from MATLAB code that uses the `coder.ceval`, `coder.extrinsic` and `coder.opaque` functions. |

| Demo... | Shows How You Can... |
|---|---|
| "Atoms" Simulation | Generate a standalone C library and executable from MATLAB code using a code generation configuration object to enable dynamic memory allocation |
| Replacing Math Functions and Operators | Use target function libraries (TFLs) to replace operators and functions in the generated code<br><br>**Note** To run this demo, you need Embedded Coder software. |
| Kalman Filter | • Generate a standalone C library from a MATLAB version of a Kalman filter<br>• Accelerate the Kalman filter algorithm by generating a MEX function |

## Functionality Being Removed in a Future Version

**Compatibility Considerations: Yes**

This function will be removed in a future version of MATLAB Coder software.

| Function Name | What Happens When You Use This Function? | Compatibility Considerations |
|---|---|---|
| `emlc` | Still runs in R2011a | None |

## Function Elements Being Removed in a Future Release

**Compatibility Considerations: Yes**

| Function or Element Name | What Happens When You Use the Function or Element? | Use This Element Instead |
|---|---|---|
| `%#eml` | Still runs | `%#codegen` |
| `eml.allowpcode` | Still runs | `coder.allowpcode` |
| `eml.ceval` | Still runs | `coder.ceval` |
| `eml.cstructname` | Still runs | `coder.cstructname` |
| `eml.extrinsic` | Still runs | `coder.extrinsic` |
| `eml.inline` | Still runs | `coder.inline` |
| `eml.nullcopy` | Still runs | `coder.nullcopy` |
| `eml.opaque` | Still runs | `coder.opaque` |
| `eml.ref` | Still runs | `coder.ref` |
| `eml.rref` | Still runs | `coder.rref` |
| `eml.target` | Still runs | `coder.target` |
| `eml.unroll` | Still runs | `coder.unroll` |
| `eml.varsize` | Still runs | `coder.varsize` |
| `eml.wref` | Still runs | `coder.wref` |

# Check bug reports for issues and fixes

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at `www.mathworks.com/support/bugreports/`. Use the Saved Searches and Watched Bugs tool with the search phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

## Search R2011a Bug Reports

Known Bugs for Incorrect Code Generation:
`www.mathworks.com/support/bugreports/?product=ALL&release=R2011a&keyword=Incorrect+Code+Generation`

All Known Bugs for This Product:
`www.mathworks.com/support/bugreports/?release=R2011a&product=ME`