

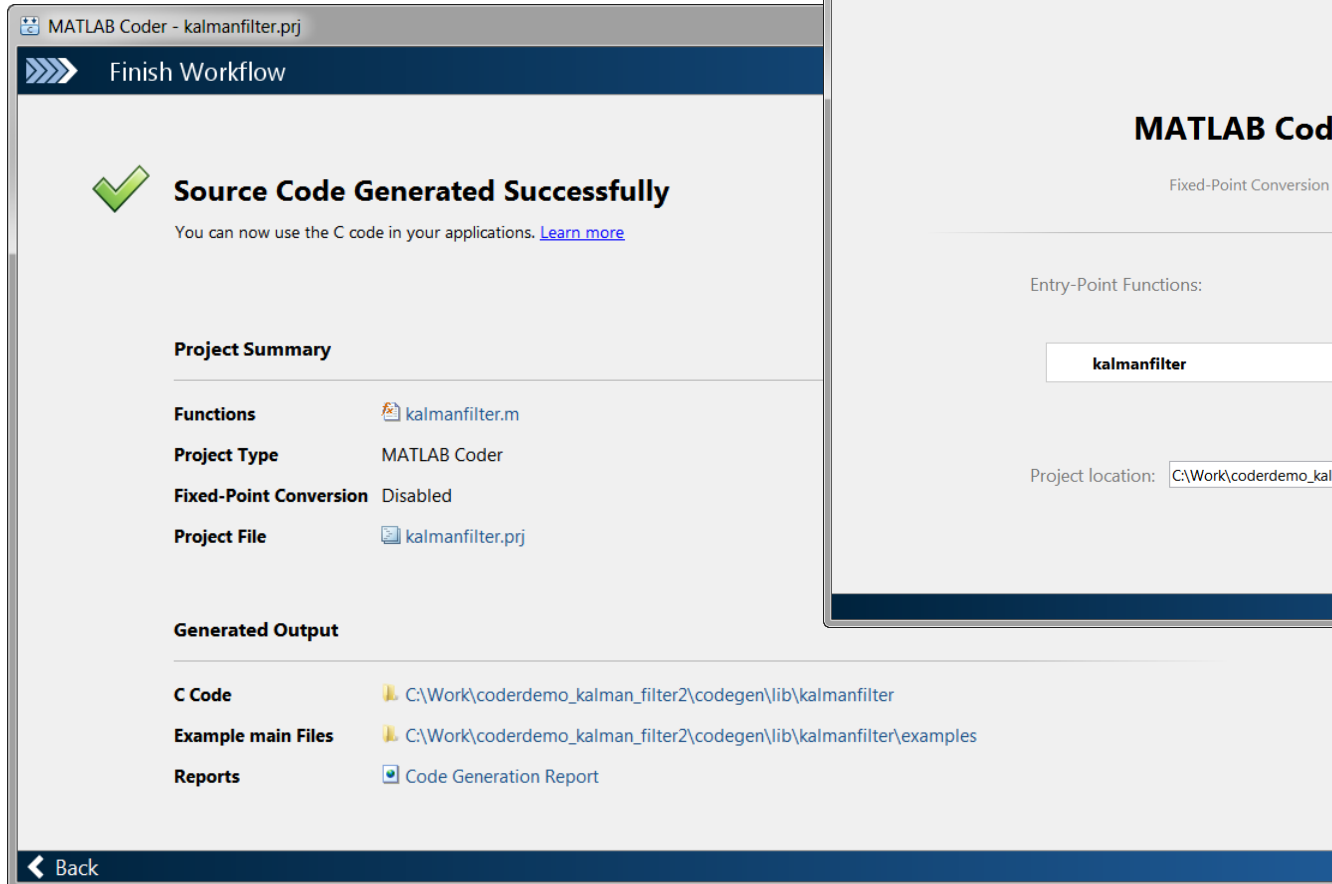
# Latest Features in MATLAB Coder

March 2015

**R2015a**


# Improved MATLAB Coder App with Integrated Editor and Simplified Workflow

New user interface simplifies code generation workflow





**MATLAB Coder - kalmanfilter.prj**




**Finish Workflow**


 **Source Code Generated Successfully**  
You can now use the C code in your applications. [Learn more](#)

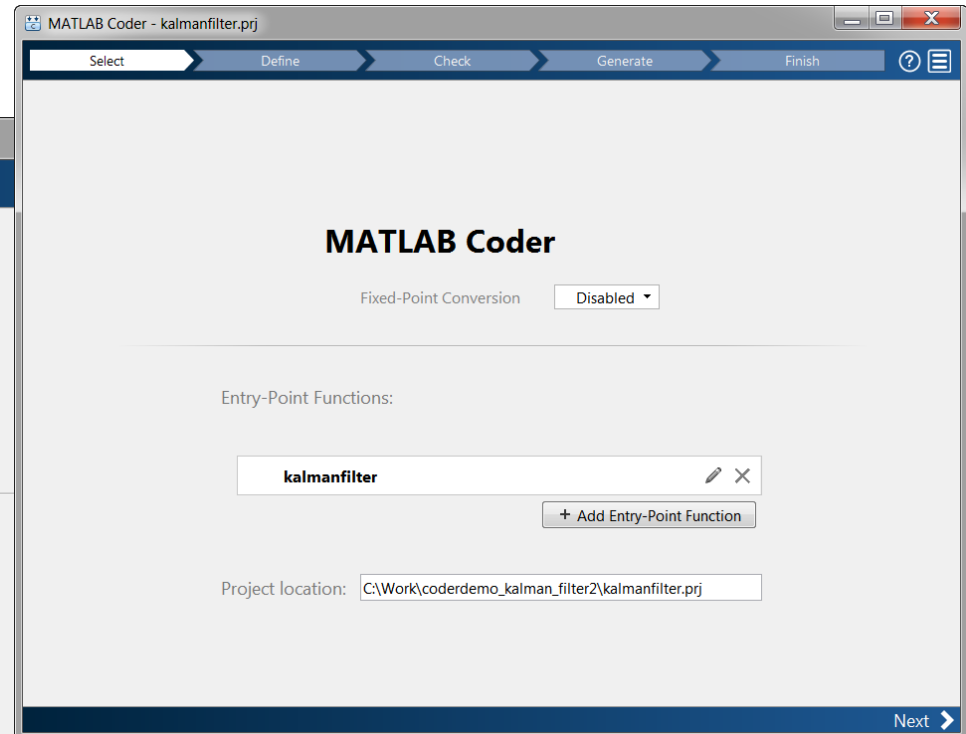
**Project Summary**

<b>Functions</b>	 kalmanfilter.m
<b>Project Type</b>	MATLAB Coder
<b>Fixed-Point Conversion</b>	Disabled
<b>Project File</b>	 kalmanfilter.prj

**Generated Output**

<b>C Code</b>	 C:\Work\coderdemo_kalman_filter2\codegen\lib\kalmanfilter
<b>Example main Files</b>	 C:\Work\coderdemo_kalman_filter2\codegen\lib\kalmanfilter\examples
<b>Reports</b>	 Code Generation Report



 Back




**MATLAB Coder**


Fixed-Point Conversion

Entry-Point Functions:

**kalmanfilter**  

 Add Entry-Point Function

Project location: C:\Work\coderdemo\_kalman\_filter2\kalmanfilter.prj

Next 

# Generation of Example C/C++ Main for Integration of Generated Code into an Application

**Use example main as template to integrate generated code into your application**

- Shows include files and how to initialize, call, and terminate generated code

```
32 /* Include Files */
33 #include "rt_nonfinite.h"
34 #include "kalmanfilter.h"
35 #include "main.h"
36 #include "kalmanfilter_terminate.h"
37 #include "kalmanfilter_initialize.h"
38
```

```
92 int main(int argc, const char * const argv[])
93 {
94     (void) argc;
95     (void) argv;
96
97     /* Initialize the application.
98        You do not need to do this more than one time. */
99     kalmanfilter_initialize();
100
101     /* Invoke the entry-point functions.
102        You can call entry-point functions multiple times. */
103     main_kalmanfilter();
104
105     /* Terminate the application.
106        You do not need to do this more than one time. */
107     kalmanfilter_terminate();
108     return 0;
109 }
110
```

# More Efficient Generated Code for Logical Indexing

**Generate faster code that uses less memory for logical array indexing**

```
resetAboveThreshold.m x +
1 function x = resetAboveThreshold(x,N)
2   assert(all(size(x) == [1 100]))
3   x(x>N) = 0;
```

```
19 void resetAboveThreshold(double x[100], double N)
20 {
21     int i;
22     for (i = 0; i < 100; i++) {
23         if (x[i] > N) {
24             x[i] = 0.0;
25         }
26     }
27 }
28
```

## Additional Code Generation Support

**Use 43 functions and System objects in MATLAB, Communications System Toolbox, Computer Vision System Toolbox, DSP System Toolbox, Image Processing Toolbox, Phased Array System Toolbox, and Statistics and Machine Learning Toolbox**

<code>bandwidth</code>	<code>cummax</code>	<code>opticalFlowLKDoG</code>
<code>betafit</code>	<code>cummin</code>	<code>pca</code>
<code>betalike</code>	<code>dsp.HighpassFilter</code>	<code>pearsrnd</code>
<code>bweuler</code>	<code>dsp.LowpassFilter</code>	<code>phased.MFSKWaveform</code>
<code>bweuler</code>	<code>extrinsics</code>	<code>phased.UCA</code>
<code>bwlabel</code>	<code>iirparameq</code>	<code>pilotcalib</code>
<code>bwperim</code>	<code>isbanded</code>	<code>reconstructScene</code>
<code>cameraMatrix</code>	<code>isdiag</code>	<code>rectifyStereoImages</code>
<code>cameraParameters</code>	<code>istril</code>	<code>regionprops</code>
<code>comm.CarrierSynchronizer</code>	<code>istriu</code>	<code>stereoParameters</code>
<code>comm.FMBroadcastDemodulator</code>	<code>lsqnonneg</code>	<code>triangulate</code>
<code>comm.FMBroadcastModulator</code>	<code>opticalFlow</code>	<code>undistortImage</code>
<code>comm.FMDemodulator</code>	<code>opticalFlowHS</code>	<code>vision.DeployableVideoPlayer</code>
<code>comm.FMModulator</code>	<code>opticalFlowLK</code>	<code>watershed</code>
<code>comm.SymbolSynchronizer</code>		

# For Use with Embedded Coder

# Indent Style and Size Control for Generated C/C++ Code

## Specify K&R indent style or Allman indent style

- K&R style places opening brace of a control statement on the same line as the control statement line as the control statement
- Allman style places the opening brace on its own line at the same indentation level as the control statement

```
void addone(const double x[6], double z[6])
{
    int i0;
    for (i0 = 0; i0 < 6; i0++) {
        z[i0] = x[i0] + 1.0;
    }
}
```

```
void addone(const double x[6], double z[6])
{
    int i0;
    for (i0 = 0; i0 < 6; i0++)
    {
        z[i0] = x[i0] + 1.0;
    }
}
```

# Improved MISRA-C Compliance for Bitwise Operations on Signed Integers and Type Cast

Increase likelihood of compliance with MISRA C®

- Specify multiplication by powers of two that reduces rule 12.7 violations

~~`i <<= 3;`~~

`i = i * 8;`

- Specify data type casts that reduces 10.1, 10.2, 10.3, and 10.4 violations

```
short addone(short x)
{
    int i0;
    i0 = x + 1;
    if (i0 > 32767) {
        i0 = 32767;
    }

    return (short)i0;
}
```

```
short addone(short x)
{
    int i0;
    i0 = (int)x + (int)1;
    if (i0 > (int)32767) {
        i0 = (int)32767;
    }

    return (short)i0;
}
```

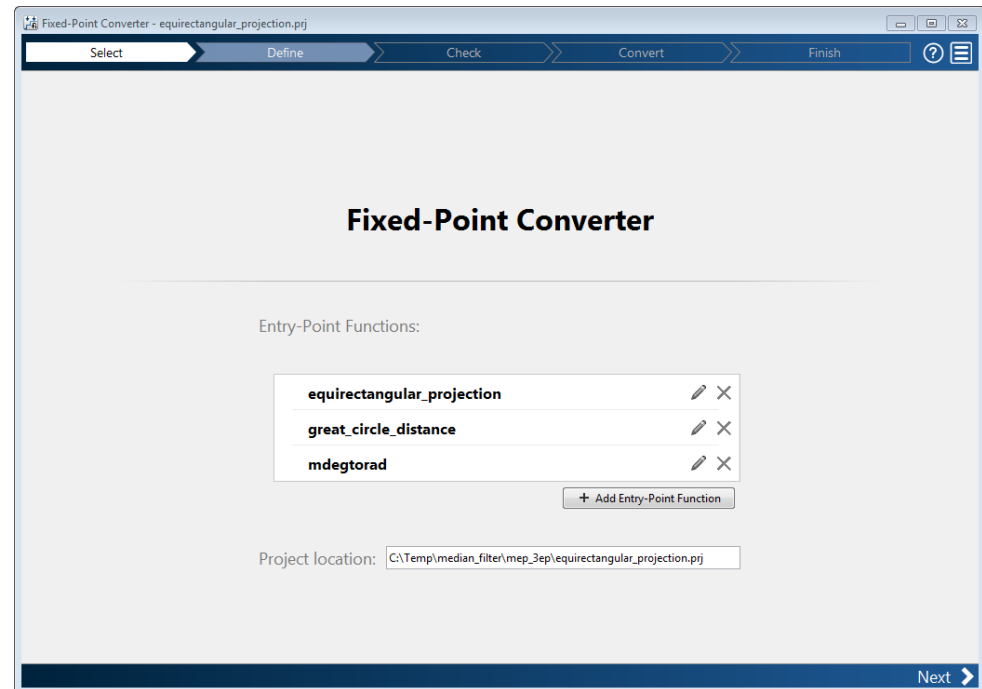


# For Use with Fixed-Point Designer

# Support for Projects with Multiple Entry-Point Functions

## Generate fixed-point code for multiple entry point functions

- Specify multiple entry-point functions in a Fixed-Point Converter app project.
- Generate fixed-point C/C++ libraries using MATLAB Coder.
- Perform conversion with multiple entry-point functions, which facilitates integration with larger applications.

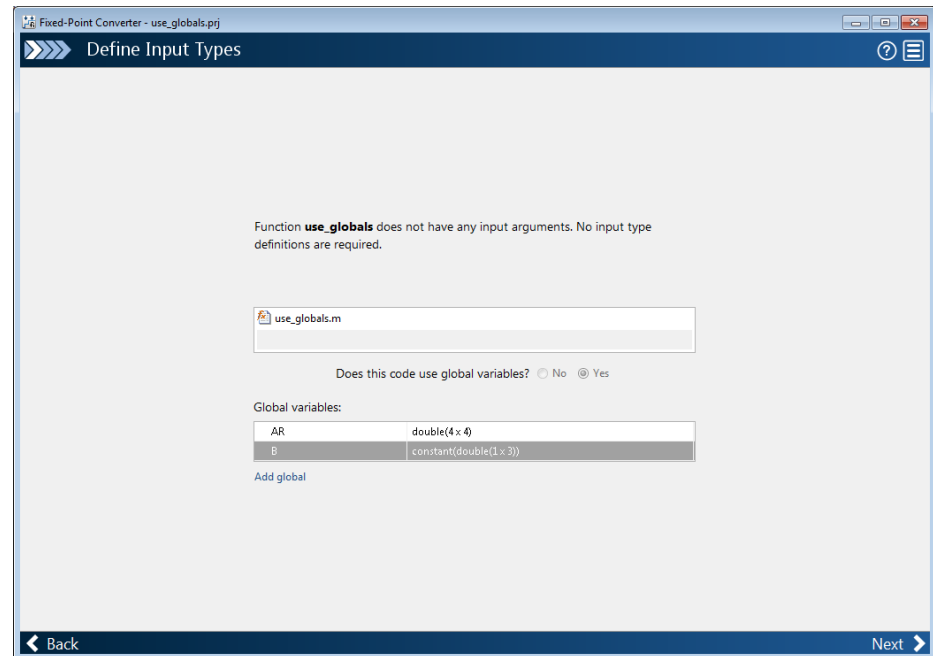


# Support for Global Variables

## Specify global variables in the Fixed-Point Converter app workflow

- Algorithms containing global variables can be converted without modifying your code.
- Ranges for globals are synchronized across functions.
- Constant globals used instead of passing constants to functions.
- Synchronize globals between the testbench and generated fixed-point code during numerical verification.

```
function y = use_globals()
%#codegen
% Turn off inlining to make
% generated code easier to read
coder.inline('never');
% Declare AR and B as global variables
global AR;
global B;
AR(1) = B(1);
y = AR * 2;
```



Fixed-Point Converter - use\_globals.m

Define Input Types

Function **use\_globals** does not have any input arguments. No input type definitions are required.

use\_globals.m

Does this code use global variables? ☐ No ☒ Yes

Global variables:

AR	double(4 x 4)
B	constant(double(1 x 3))

Add global

Back Next

# Smart Conversion of Dead and Constant Folded Code

Fixed-Point Converter app detects constant folded and dead code to reduce translation errors

- Augments test files to exercise the algorithm adequately
- Inline comments in fixed-point MATLAB code to mark dead and untranslated regions
- Displays code execution information in generated conversion report and as color-code bars in editor view
- Supports command-line workflow



Generated on 2015-02-24 18:54:50

The following table shows fixed point instrumentation results

## Fixed-Point Report *mlhdlc\_dti*

Simulation Coverage	Code
100%	function [y, is_clipped] = mlhdlc_dti(u_in, init_val, gain_val, upper_limit, lower_limit) % Discrete Time Integrator in MATLAB Function block
	% Forward Euler method, also known as Forward Rectangular, % or left-hand approximation. The resulting expression for the % output of the block at step n is % % y(n) = y(n-1) + K * u(n-1) % % Setup % % numeric type to clip the accumulator value after each addition % variable to hold state between consecutive calls to this block persistent u_state; if isempty(u_state) u_state = init_val; end
Once	
100%	% clip flag status positive_sat_occurred = 1; negative_sat_occurred = -1; no_sat_occurred = 0;

Code execution serialized to html report