

DSP System Toolbox™ Release Notes



MATLAB® & SIMULINK®

R2017b



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

DSP System Toolbox™ Release Notes

© COPYRIGHT 2012–2017 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Improved Spectrum Analyzer: Analyze signals in the frequency domain using polyphase FFT filter banks, custom windows, dBFS units, and a spectral mask panel	1-2
Zoom FFT: Compute fast Fourier transform (FFT) of a frequency subband at high resolution	1-3
Frequency-Domain FIR Filter: Convolve long sequences while balancing latency and execution efficiency	1-3
Multitap Fractional Delay: Delay signals by multiple sample period values concurrently using variable fractional delay	1-3
Minimum Resource FFT/IFFT: Reduce resource usage with the Burst Radix 2 architecture of the HDL Optimized FFT (requires HDL Coder for code generation)	1-3
Logic Analyzer Improvements: Triggers and bus signal names	1-4
Enhancements to the dsp.Channelizer System object	1-4
Automatic Port Creation: Add inports to scope blocks when routing signals	1-4
Improvements to interactive legend in scope blocks	1-4
Array Plot Improvements: Support for scalar and variable-size inputs, axis scaling at the command line	1-5

dsp.BlockLMSFilter System object supports code generation	1-5
Functionality being removed	1-5
Removal of Overlap-Add FFT Filter block and Overlap-Save FFT Filter block	1-5
Removal of sample-based processing mode from the DSP System Toolbox System objects	1-5
Removal of adaptfilt objects	1-6
Removal of qfft and qformat functions	1-8
Removal of HDL Minimum Resource FFT block	1-9
Removal of Streaming Radix 2 architecture in HDL-optimized FFT blocks and System objects	1-9

R2017a

Improved Spectrum Analyzer: Analyze signals in the frequency domain using additional units, dual visualization, and mask compliance output	2-2
Unified interface for dsp.LogicAnalyzer: Visualize, measure, and analyze signal transitions in MATLAB using the same interface as the Simulink Logic Analyzer	2-3
Channelizer and Channel Synthesizer Blocks: Analyze and synthesize narrow subbands of a broadband signal using a polyphase FFT filter bank in Simulink	2-3
Asynchronous Buffering: Exchange signals at different rates and array sizes with the dsp.AsyncBuffer System object	2-3
HDL Optimized Filters: Model and generate optimized hardware implementations for FIR filters and polyphase filter banks (requires HDL Coder for code generation) ...	2-4
Discrete FIR Filter	2-4
Polyphase Filter Bank	2-4
Frame Input Support for FIR Decimation	2-4

Remove outliers from streaming signals in MATLAB and Simulink using Hampel filter	2-5
Spectral estimation using filter bank in Simulink	2-5
Tunable UDP port number in generated code	2-5
Filter signals using the dsp.FilterCascade System object	2-5
Use delay and scalar gain in dsp.FilterCascade System object	2-6
Cascade a dsp.FilterCascade System object	2-6
Access the complete history of LMS filter weights in MATLAB	2-6
Tab Completion: Complete parameter names and options in DSP System Toolbox System objects	2-6
Filter Builder and fdesign support IIR halfband filter System objects	2-7
Specify image file icons for MATLAB System block	2-7
Change tunable System object properties before locking	2-8
Support for Time Scope to For Each subsystems	2-8
Copy scope to clipboard	2-8
Interactive legend for scopes	2-8
Stem plot option for Time Scope block	2-8
Time Scope Block: Connect nonvirtual bus and array of buses signals	2-9
Frame-based processing changes	2-9
Input processing parameter set to Inherited	2-9
InputProcessing property set to Inherited errors	2-11
Rate options parameter set to Inherit from input	2-11
Find the histogram over parameter set to Inherited	2-11

Running difference parameter set to Inherit from input	2-12
Save 2-D signals as parameter set to Inherit from input	2-12
Treat Mx1 and unoriented sample-based signals as parameter removed	2-12
Sample-based processing parameter removed	2-13
Functionality being removed	2-13
Running Mode in Statistics Objects and Blocks	2-13
Audio device recorder and player objects	2-14
Radix 2 architecture of HDL-optimized FFT blocks and System objects	2-15

R2016b

Logic Analyzer: Visualize, measure, and analyze transitions and states over time for Simulink signals	3-2
Spectral Mask: Compare a signal spectrum to a spectral mask using Spectrum Analyzer	3-2
Channelizer and Channel Synthesizer: Analyze and synthesize narrow subbands of a broadband signal using a polyphase FFT filter bank	3-2
Moving Statistics: Measure descriptive statistics on streaming signals in MATLAB and Simulink	3-2
Gigasample per Second (GSPS) Signal Processing: Increase the throughput of HDL code generated from Discrete FIR Filter and Integer Delay blocks using frame input	3-3
Stream signals to and from binary files	3-3
Compute LMS adaptive filter weights using LMS Update block	3-3
Allpass Filter block	3-4

Specify coefficients in Farrow Rate Converter block and System object	3-4
Spectral estimation using filter banks	3-4
High-throughput polyphase filter bank for HDL example ...	3-4
Bit-reversed input order for HDL-optimized FFT	3-4
HDL code generation for reset port on Discrete FIR Filter	3-5
Compiler support for System object scopes	3-5
Custom X-axis data in Array Plot	3-5
Set legend strings and autoscaling programmatically in Time Scope	3-5
Simpler way to call System objects	3-5
System objects support for additional inputs, global variables, and enumeration data types	3-6
Functionality being removed	3-6
Removal of sample mode from the DSP System Toolbox System objects	3-6
Digital Filter block and System object	3-7
Removal of adaptfilt objects	3-8
Cell array support removal for dsp.AllpassFilter coefficients	3-9
Inherited option removed from the input processing parameter	3-9
Frame status parameter removed from the Check Signal Attributes block	3-10
qfft object errors	3-10
dspstartup removed	3-10

DSP Unfolding for Mac: Generate multithreaded MEX files from MATLAB functions on Mac OS X	4-2
Faster FIR and Biquad Filters: Run faster simulations for system models that include FIR and biquad filters	4-2
Fixed-Point Farrow Rate Converter: Design and simulate Farrow rate conversion filters using fixed-point data types	4-2
Gigasample per Second (GSPS) Signal Processing: Increase throughput of HDL-optimized FFT and IFFT algorithms using frame input	4-2
HDL Optimizations for Biquad Filter: Reduce critical path or area when generating HDL from a subsystem that includes a Biquad Filter block	4-3
Differentiate a signal using the dsp.Differentiator System object and Differentiator block	4-3
Play audio data using the audioDeviceWriter System object and Audio Device Writer block	4-3
Specify coefficients in IIR Halfband Interpolator and IIR Halfband Decimator Blocks and System objects	4-4
Customize the data limits of the Matrix Viewer block	4-4
Code generation for wave digital filter structure in dsp.AllpassFilter System object	4-4
Generate coefficients for multirate filters	4-5
Select the color of the noise in dsp.ColoredNoise System object	4-5
Full-precision setting for product data type of Biquad Filter	4-5

Code generation for Subband Analysis and Subband Synthesis Filters	4-6
Enhancements to Variable Fractional Delay	4-6
Multiple inputs for Spectrum Analyzer	4-6
Additional axes for Time Scope	4-6
Set legend programmatically in Array Plot	4-7
System object property display	4-7
System object enhancements to MATLAB System block	4-7
Enhanced System Object Development with MATLAB Editor	4-8
Functionality being removed	4-8

R2015b

DSP Unfolding: Generate a multi-threaded MEX File from a MATLAB function	5-2
HDL Optimizations for Discrete FIR Filter: Implement FIR filters in hardware at higher frequencies or using fewer resources	5-2
Array Plot Block: Visualize array and vector data	5-2
Additional Multirate Filters: Design Halfband, CIC compensation, and HDL-optimized FIR rate conversion filters	5-2
Conversion Filter Blocks: Convert the rate of signals in Simulink models	5-3

Implement FIR and IIR filters in Simulink, using the Lowpass Filter and Highpass Filter blocks	5-3
Estimate power spectrum and power spectral density using the Spectrum Estimator block	5-4
Automatic selection of filter coefficients for FIR Interpolation, FIR Decimation, and FIR Rate Conversion blocks	5-4
Visualize the frequency response of the underlying filters in the DSP System Toolbox blocks	5-4
Specify the window length and window overlap in Cross-Spectrum Estimator and Discrete Transfer Function Estimator blocks	5-5
Select the color of the noise in Colored Noise block	5-5
New functionality added to the dsp.SpectrumEstimator System object	5-5
Generate C code from dsp.AllpassFilter and import the System object into Simulink using the MATLAB System block	5-6
dsp.CICDecimator and dsp.CICInterpolator System objects support single and double data types	5-7
Frame-based signal logging in structure formats in Time Scope block	5-7
Scientific notation in Time Scope	5-7
Performance improvements for FFT, IFFT and notch peak filters	5-7
Floating-point support and optional valid port for HDL-optimized NCO	5-7
HDL Code Generation from filterbuilder	5-8

Simulink templates for ARM Cortex-A and ARM Cortex-M processors	5-8
ROI processing removed	5-9
Frame-based processing changes	5-9
Inherited Option Removed from the Input Processing Parameter	5-9
Sample-Based Row Vector Processing Changes	5-10
Blocks Emit Sample-Based Signals Only	5-12
Features removed, replaced and renamed	5-13
Blocks removed and replaced	5-13
Removal of adaptfilt objects	5-14
Removal of mfilt objects	5-14
System Object Propagation Mixin Methods Renamed	5-15

R2015a

Audio Latency Reduction: Significantly reduce latency for audio hardware I/O in MATLAB and Simulink	6-2
Filter Design Enhancements: Design high-order IIR parametric EQ filter, variable bandwidth FIR and IIR filters, Digital Down-Converter and Digital Up-Converter blocks	6-2
DSP Simulink Model Templates: Configure the Simulink environment for digital signal processing models	6-3
Streaming Scope Improvements: Plot in stem mode, access log x-axis scaling, customize sample rate, and use infinite data support	6-3
Library for HDL Supported DSP Blocks: Find all blocks that support HDL	6-4

C Code Generation of DSP Algorithms for ARM Cortex-A and Cortex-M processors: Generate optimized and faster performing C code using Embedded Coder	6-4
Performance Improvements	6-5
Updated Time Scope block toolbar and menus	6-5
Specify block filter characteristics through System objects	6-5
Discrete Transfer Function Estimator block	6-6
Specify filter coefficients as an input to the FIR Decimation block	6-6
Enhanced code generation for CIC Decimation and CIC Interpolation filter blocks	6-7
HDL support for ‘inherit via internal rule’ data type setting on FIR Decimation and Interpolation blocks	6-7
Improvements for creating System objects	6-7
Min/Max logging instrumentation for float-to-fixed-point conversion of DSP System objects	6-7
Provide variable-size input to the Delay System object	6-8
Estimate output coherence of Transfer Function Estimator System object	6-8
Specify filter coefficients as an input to the FIR Decimator System object	6-8
Bit growth to avoid overflow in HDL-optimized FFT and IFFT	6-9
Fixed-point support for FIR Half-band Interpolator and FIR Half-band Decimator System objects	6-9
Updated cost method for filter System objects	6-9

Frame-based processing	6-9
Input processing parameter set to Inherited	6-10
Rate options parameter set to Inherit from input	6-24
Treat Mx1 and unoriented sample-based signals as parameter set to M channels	6-25
Save 2-D signals as parameter set to Inherit from input	6-25
Find the histogram over parameter set to Inherited	6-26
Sample-based processing parameter set to Pass through	6-26
Running difference parameter set to Inherit from input	6-27
Features removed, replaced, and duplicated	6-28
Blocks replaced, removed, and available in additional libraries	6-28
Removal of adaptfilt objects	6-29
Functionality changed or being removed for blocks and System objects	6-30
Removal of sample mode from the DSP System Toolbox System objects	6-30
Option to specify filter coefficients from Digital Up Converter and Digital Down Converter System objects being removed	6-31
Removal of OutputDataType and OverflowAction properties for CIC Compensation Interpolator and Decimator System objects	6-32

R2014b

Optimized C code generation for ARM Cortex-A Ne10 library from MATLAB and Simulink with DSP System Toolbox Support Package for ARM Cortex-A Processors	7-2
System objects for DSP System Toolbox Support Package for ARM Cortex-M Processors	7-3
Fixed-point support for Biquad Filter on DSP System Toolbox Support Package for ARM Cortex-M Processors	7-3

Multirate filters: Sample and Farrow Rate Converter, CIC Compensation Interpolator/Decimator, and FIR Halfband Interpolator/Decimator System objects	7-3
Tunable coefficients and variable-size input available on FIR Interpolator System object and block	7-4
Variable-size input available on FIR Decimator System object and block	7-4
Min/Max logging instrumentation for float-to-fixed-point conversion of commonly used DSP System objects, including Biquad Filter, FIR Filter, and FIR Rate Converter	7-4
HDL-optimized FFT and IFFT System objects and HDL-optimized Complex to Magnitude-Angle System object and block	7-5
Real input, bit-reversed output, reset input available on HDL-optimized FFT and IFFT	7-5
Option to synthesize lookup table to ROM available on HDL-optimized FFT and IFFT blocks	7-5
Reduced latency of HDL-optimized FFT and IFFT	7-6
CIC algorithm and HDL code generation for DC Blocker ...	7-6
dsp.FilterCascade System object	7-6
Phase Extractor block and dsp.PhaseExtractor System object	7-6
Overflow and underflow reporting on audio device blocks and System objects	7-6
Unsigned input data type in dsp.CICDecimator and dsp.CICInterpolator System Objects	7-7
Logic Analyzer support for vector, enumerated, and complex inputs	7-7

System object support in Simulink For Each Subsystem	7-7
Getting Started Tutorials	7-7
Functionality being removed or replaced for blocks and System objects	7-8
Persistence mode in Vector Scope	7-16
Code generation for additional DSP System Toolbox System objects	7-16
Tunable amplitude on dsp.SineWave	7-17

R2014a

Up to four-times faster FIR filter simulation in MATLAB System object and Simulink block	8-2
Optimized C code generation for ARM Cortex-M processors from System objects with MATLAB Coder and Embedded Coder	8-2
Notch/peak filter and parametric equalizer filter System objects in MATLAB	8-3
Variable bandwidth FIR and IIR filter System objects in MATLAB	8-3
Pink/Colored noise generation System object in MATLAB . .	8-3
HDL optimized FFT and IFFT Simulink blocks	8-3
Fixed-point data type support for FIR filter, in ARM Cortex-M support package	8-3
Choice of wrapping or truncating input of FFT, IFFT, and Magnitude FFT in MATLAB and Simulink	8-4

Variable-size input for biquad and LMS filters in MATLAB and Simulink	8-4
More flexible control of dsp.LMSFilter System object fixed-point settings	8-4
DC blocker System object and Simulink block	8-4
dsp.DigitalDownConverter and dsp.DigitalUpConverter now support C code generation	8-5
The isDone method of dsp.AudioFileReader honors PlayCount	8-5
M4A replaced by MPEG4 in dsp.AudioFileWriter	8-5
Spectrogram cursors and CCDF plots in the spectrum analyzer	8-5
Changed dsp.SpectrumAnalyzer property names	8-5
Conversion to/from allpass from/to wave digital filter	8-6
Transfer function estimation in Simulink	8-6
Updates to the Time Scope	8-6
Changed dsp.TimeScope property names	8-7
Time Scope automatically switches to block-based sample time	8-7
dsp.LogicAnalyzer channel selection	8-7
System object templates	8-7
System objects infer number of inputs and outputs from stepImpl method	8-8
System objects setupImpl method enhancement	8-8
System objects infoImpl method allows variable inputs	8-8

System objects base class renamed to matlab.System	8-8
System objects Propagates mixin methods	8-8
Code generation support for additional functions	8-9

R2013b

Support Package for ARM Cortex-M Processors	9-2
Channel and distortion measurement, cursors, and spectrogram visualization using Spectrum Analyzer in MATLAB and Simulink	9-2
Channel mapping for multichannel audio devices in MATLAB and Simulink	9-3
Variable-size support for FIR and Allpole filters in MATLAB and Simulink	9-3
Estimation of Power Spectrum, Cross Power Spectrum, and Transfer Function for streaming data in MATLAB	9-3
Data logging and archiving using Time Scope in Simulink	9-4
MIDI control interface support in MATLAB	9-4
Integer support on the output port of the MIDI Controls block	9-4
Kalman filter	9-4
Adaptive filters using Lattice, Fast Transversal, Filtered-X LMS, and Frequency Domain algorithms in MATLAB	9-5
Coupled allpass filter	9-5

Functionality being removed or changed	9-6
Migrate away from fdesign.pulseshaping	9-6
Configuration dialog added to Logic Analyzer	9-7
Complex trigger support in Time Scope	9-7
Default color changes for Array Plot, Time Scope, and Spectrum Analyzer	9-7
MATLAB System Block to include System objects in Simulink models	9-7
Restrictions on modifying properties in System object Impl methods	9-7
System objects matlab.system.System warnings	9-9
Removing HDL Support for NCO Block	9-9

R2013a

Allpass Filter System object	10-2
Adaptive filter System objects using RLS and Affine Projection Filter	10-2
Logic Analyzer System object	10-2
Audio System object support for tunability, variable frame size, variable number of channels, and writing MPEG-4 AAC	10-3
Array Plot System object for displaying vectors or arrays in 2-D and Spectrum Analyzer block with enhanced controls and features such as peak finder	10-4

Time Scope block with triggering and peak finder	
features	10-10
Triggers Panel	10-11
Peak Finder Features	10-11
Panning Capability	10-11
Programmatic Access	10-11
Scale Axes Limits After 10 Updates	10-12
Change of the default for audio hardware API on Linux ..	10-12
Change of the default for audio file formats in multimedia blocks and audio file reader and writer System	
objects	10-12
Change of property default in the audio file reader System object	10-12
Removal of the signalblks package	10-12
Scope Snapshot display of additional scopes in Simulink Report Generator	10-13
Unoriented vector treated as column vector in the Biquad Filter	10-13
NCO HDL Optimized block	10-13
HDLNCO System object	10-13
HDL code generation for NCO HDL Optimized block and System object	10-14
Support for nonpersistent System objects	10-14
New method for action when System object input size changes	10-14
Scaled double data type support for System objects	10-14

SpectrumAnalyzer System object	11-2
Cross-platform support for reading and writing WAV, FLAC, OGG, MP3 (read only), MP4 (read only), and M4a (read only)	11-3
Support for code generation for CICDecimator and CICInterpolator System objects	11-3
Support for HDL code generation for multichannel Discrete FIR Filter block	11-4
Time Scope enhancements, including new cursors, embedded simulation controls, and External and Rapid Accelerator modes	11-4
Cursor measurements panel	11-4
Additional embedded simulation controls	11-5
Support for external mode and rapid accelerator mode	11-5
Properties dialog box	11-6
Axes Maximization	11-7
Automatic calculation of Time Span	11-7
ReduceUpdates property	11-7
Support for conditional subsystems	11-8
Source and sink blocks being replaced	11-8
Discrete IIRFilter and AllpoleFilter System objects	11-9
Support for MATLAB Compiler for CICDecimator and CICInterpolator System objects	11-9
Code generation support for SignalSource System object	11-9
Behavior change of locked System objects for loading, saving, and cloning	11-10
Behavior change of statistics blocks for variable-size inputs	11-10

Simulation state save and restore for additional blocks . .	11-11
For Each subsystem support for additional blocks	11-12
Multi-instance model referencing support for additional blocks	11-12
Expanded analysis support for filter System objects	11-12
Removal of the signalblks package	11-12
Discrete filter block visible in DSP library	11-13
System object tunable parameter support in code generation	11-13
save and load methods for System objects	11-13
Save and restore SimState not supported for System objects	11-13
Map integer delay to RAM on Delay block	11-14
HDL support for System objects	11-14
HDL resource sharing for Biquad Filter block	11-14

R2012a

Frame-Based Processing	12-2
Inherited Option of the Input Processing Parameter Now Warns	12-2
Logging Frame-Based Signals in Simulink	12-3
Model Reference and Using slupdate	12-4
Removing Mixed Frameness Support for Bus Signals on Unit Delay and Delay	12-4
Audio Output Sampling Mode Added to the From Multimedia File Block	12-5

System Object Enhancements	12-5
Code Generation for System Objects	12-5
New MAT-File Reader and Writer System Objects	12-5
New System Object Option on File Menu	12-6
Variable-Size Input Support for System Objects	12-6
Data Type Support for System Objects	12-6
New Property Attribute to Define States	12-6
New Methods to Validate Properties and Get States from System Objects	12-6
matlab.system.System changed to matlab.System	12-6
Time Scope Enhancements	12-7
Time Domain Measurements in Time Scope	12-7
Multiple Display Support in Time Scope	12-7
Style Dialog Box in Time Scope	12-8
Sampled Data as Stairs in Time Scope	12-8
Complex Data Support in Time Scope	12-9
Additional Time Scope Enhancements	12-9
ASIO Support in To/From Audio Device Blocks and Objects	12-10
Video Processing Enabled for the DSP System Toolbox Multimedia File Blocks	12-10
System Objects Integrated into Filter Design Workflow ..	12-10
Integration of System Objects into Filter Design via fdesign, FDATool, and Filterbuilder	12-10
Convert dfilt and mfilt Filter Objects to System Objects ...	12-11
Filter Analysis and Conversion Methods for System Object Filters	12-11
New Measurement Workflow	12-11
Measurements for Bilevel Pulse Waveforms	12-11
System Objects for Peak-to-RMS and Peak-to-Peak Measurements	12-12
Discrete FIR Filter System Object	12-12
Inverse Dirichlet Sinc-Shaped Passband Design Added to Constrained FIR Equiripple Filter	12-12

Code Generation Support Added to FIR Decimator System Object	12-13
Filter Block Enhancements	12-13
IC/Coefficient Parameter Ports in the Simulink Discrete Filter and Discrete Transfer Function	12-13
Reset Port for Resetting Filter State in Filter Blocks	12-13
Discrete FIR Filter Block Coefficient Port Changes	12-13
Statistics Blocks and Objects Warning for Region of Interest Processing	12-14
New and Updated Demos	12-14

R2011b

Frame-Based Processing	13-2
General Product-Wide Changes	13-2
Logging Signals in Simulink	13-4
Triggered to Workspace	13-4
Digital Filter Design Block	13-5
Filterbuilder, FDATool and the Filter Realization Wizard Block	13-6
Changes to Row Vector Processing for dsp.Convolver, dsp.CrossCorrelator, and dsp.Interpolator System Objects	13-6
Custom System Objects	13-7
New Allpole Filter Block	13-7
New Audio Weighting Filter Functionality	13-7
Time Scope Enhancements	13-7
New Arbitrary Group Delay Design Support	13-8

Arbitrary Magnitude Responses Now Support Minimum Order and Minimum/Maximum Phase Equiripple Design Options	13-8
Support for Constrained Band Equiripple Designs in MATLAB and Simulink	13-8
New Sinc Frequency Factor and Sinc Power Design Options for Inverse Sinc Filters	13-9
New Inverse Sinc Highpass Filter Designs	13-9
Filterbuilder and dspfdesign Library Blocks Now Support Different Numerator and Denominator Orders for IIR Filters	13-9
New Stopband Shape and Stopband Decay Design Options for Equiripple Highpass Filter Designs	13-10
FFTW Library Support for Non-Power-of-Two Transform Length	13-10
MATLAB Compiler Support for dsp.DigitalDownConverter and dsp.DigitalUpConverter	13-10
Complex Input Support for dsp.DigitalDownConverter ..	13-10
getFilters Method of dsp.DigitalDownConverter and dsp.DigitalUpConverter Now Return Actual Fixed-Point Settings	13-10
dsp.SineWave and dsp.BiquadFilter Properties Not Tunable	13-11
System Object DataType and CustomDataType Properties Changes	13-11
System Objects Variable-Size Input Dimensions	13-12
Conversion of Error and Warning Message Identifiers ...	13-12
New and Updated Demos	13-13

Product Restructuring	14-2
Frame-Based Processing	14-2
General Product-Wide Changes	14-2
Blocks with a New Input Processing Parameter	14-4
Changes to the Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT Blocks	14-6
Difference Block Changes	14-7
Signal To Workspace Block Changes	14-7
Spectrum Scope Block Changes	14-8
Sample-Based Row Vector Processing Changes	14-8
New Function for Changing the System Object Package Name from signalblks to dsp	14-11
New Discrete FIR Filter Block	14-11
New Printing Capability from the Time Scope Block	14-11
Improved Display Updates for the Time Scope Block and System Object	14-12
New Implementation Options Added to Blocks in the Filter Designs Library	14-12
New dsp.DigitalDownConverter and dsp.DigitalUpConverter System Objects	14-13
Improved Performance of FFT Implementation with FFTW library	14-13
Variable-Size Support for System Objects	14-13
System Objects FullPrecisionOverride Property Added	14-14

'Internal rule' System Object Property Value Changed to 'Full precision'	14-15
MATLAB Compiler Support for System Objects	14-15
Viewing System Objects in the MATLAB Variable Editor .	14-16
System Object Input and Property Warnings Changed to Errors	14-16
New and Updated Demos	14-16
Documentation Examples Renamed	14-17
Downsample Block No Longer Has Frame-Based Processing Latency for a Frame Size of One	14-17
SignalReader System Object Accepts Column Input Only	14-18
FrameBasedProcessing Property Removed from the dsp.DelayLine and dsp.Normalizer System Objects	14-18
R2010a MAT Files with System Objects Load Incorrectly .	14-18

R2017b

Version: 9.5

New Features

Bug Fixes


Compatibility Considerations

Improved Spectrum Analyzer: Analyze signals in the frequency domain using polyphase FFT filter banks, custom windows, dBFS units, and a spectral mask panel

Spectrum Analyzer blocks and `dsp.SpectrumAnalyzer` System objects have been improved.

- Analyze frequency signals using a polyphase FFT filter bank spectral estimation. The filter bank method has a lower noise floor and better frequency resolution. Filter bank also requires fewer samples per update compared to the Welch method. To use filter bank estimation in the Spectrum Analyzer block, in the Spectrum Setting panel, set **Method** to `Filter Bank`. For the System object™, use `dsp.SpectrumAnalyzer('Method','Filter Bank')`.

Two new properties allow you to control the spectral estimation method and the filter bank estimation:

- **Method** – Choose the spectral estimation method.
- **NumTapsPerBand** – Specify the number of taps per frequency band.
- Save the spectrum data plotted in the Spectrum Analyzer using two new functions:
 - `getSpectrumData` - Save the spectrum or spectrogram displayed in the Spectrum Analyzer along with simulation time, frequency vector, min hold, and max hold.
 - `isNewDataReady` – Check if spectrum is updated to avoid saving duplicate data.
- Window your spectral analysis with two new **Window** options: `Blackman-Harris` and `Custom`. For more information, see `Window` and `CustomWindow` .
- Analyze spectrum data in dBFS. When using `dBFS` `SpectrumUnits` , you can customize the full scale with the new properties `FullScale` and `FullScaleSource` .
- Customize axis scaling at the command line with the `AxesScaling` property for auto scaling and the `ColorLimits` and `YLimits` properties for hard-coded limits.
- Within the Spectrum Analyzer window, set up spectral masks and monitor how often the mask fails. In the toolbar, select the  button to show the spectral mask settings and statistics panel.

Zoom FFT: Compute fast Fourier transform (FFT) of a frequency subband at high resolution

Analyze a subband of frequencies at a high resolution using the zoom FFT algorithm in the `dsp.ZoomFFT` System object and Zoom FFT block.

Frequency-Domain FIR Filter: Convolve long sequences while balancing latency and execution efficiency

Using the `dsp.FrequencyDomainFIRFilter` System object and the Frequency-Domain FIR Filter block, you can filter a streaming input signal using FFT-based filtering methods. To mitigate the latency for a long filter, you can partition the impulse response into shorter blocks, and apply the filtering on these blocks.

Multitap Fractional Delay: Delay signals by multiple sample period values concurrently using variable fractional delay

Concurrently delay signals by multiple fractional delay values using the `dsp.VariableFractionalDelay` System object and the Variable Fractional Delay block.

Minimum Resource FFT/IFFT: Reduce resource usage with the Burst Radix 2 architecture of the HDL Optimized FFT (requires HDL Coder for code generation)

You can now choose a minimum resource architecture for the HDL-optimized FFT blocks and System objects. To use this feature, select the `Burst Radix 2` architecture in these blocks and System objects:

- FFT HDL Optimized
- IFFT HDL Optimized
- `dsp.HDLFFT`
- `dsp.HDLIFFT`

Logic Analyzer Improvements: Triggers and bus signal names

- In the **Logic Analyzer** and `dsp.LogicAnalyzer`, you can now use triggers to display signal data when certain conditions are met. Once you attach a signal to a trigger, you can trigger on:
 - Rising or falling edges
 - Bit pattern matching
 - Less than or greater than a value
 - Equal to a value
- If you log a bus signal in the **Logic Analyzer**, you can now view the bus element names. In the Logic Analyzer Settings window, select **Display bus element names**.

Enhancements to the `dsp.Channelizer` System object

You can now compute and visualize the frequency response of an individual or a combination of the filters in the `dsp.Channelizer` System object, using the `freqz` and the `fvtool` functions. You can also obtain the bandedge frequencies and the center frequencies of the bandpass filters in the channelizer using the `bandedgeFrequencies` and the `centerFrequencies` functions. The `getFilters` function returns a matrix of filter coefficients, with each row containing the coefficients of the corresponding bandpass filter.

Automatic Port Creation: Add inports to scope blocks when routing signals

For Spectrum Analyzer and Array Plot blocks in Simulink models, dragging a line to connect another signal to the scope automatically adds a new input port. For an example, see “Build and Edit a Model in the Simulink Editor” (Simulink).

Improvements to interactive legend in scope blocks

For scope blocks and System objects, use the scope legend to toggle signal visibility. In the scope legend, click a signal name to hide the signal in the scope. To show the signal, click the signal name again. To show only one signal, right-click the signal name, which hides all other signals.

Array Plot Improvements: Support for scalar and variable-size inputs, axis scaling at the command line

For Array Plot blocks and `dsp.ArrayPlot` system objects, you can now:

- Visualize scalar or variable-sized input signals. If the signal is variable sized, the number of channels (columns) cannot change.
- Customize array plot axis scaling at the command line with the `AxesScaling` property.

`dsp.BlockLMSFilter` System object supports code generation

Generate C and C++ code from MATLAB® code that contains the `dsp.BlockLMSFilter` System object using the MATLAB Coder™.

Functionality being removed

Removal of Overlap-Add FFT Filter block and Overlap-Save FFT Filter block

Overlap-Add FFT Filter and Overlap-Save FFT Filter blocks have been replaced with the Frequency-Domain FIR Filter block. Existing instances of these blocks continue to run. For new models, use the Frequency-Domain FIR Filter block instead.

Removal of sample-based processing mode from the DSP System Toolbox System objects

The `FrameBasedProcessing` property of all DSP System objects has been removed. All the System objects listed now only work in frame-based processing mode. See “What Is Frame-Based Processing?” for more information.

- `dsp.AllpoleFilter`
- `dsp.AnalyticSignal`
- `dsp.BiquadFilter`
- `dsp.Buffer`
- `dsp.CumulativeProduct`
- `dsp.CumulativeSum`
- `dsp.Delay`

- `dsp.FIRFilter`
- `dsp.IIRFilter`
- `dsp.MatFileReader`
- `dsp.MatFileWriter`
- `dsp.Maximum`
- `dsp.Mean`
- `dsp.Minimum`
- `dsp.PeakToPeak`
- `dsp.PeakToRMS`
- `dsp.PhaseUnwrapper`
- `dsp.RMS`
- `dsp.SignalSink`
- `dsp.StandardDeviation`
- `dsp.VariableFractionalDelay`
- `dsp.VariableIntegerDelay`
- `dsp.Variance`
- `dsp.ZeroCrossingDetector`

Compatibility Considerations

In existing code, if this property is set to `true`, no further change to the input is required. If this property is set to `false` and the input is a column vector or an N -D array, reshape the input such that each column in the input is an independent channel.

Removal of `adaptfilt` objects

All `adaptfilt` objects have been removed. Use the corresponding `System` object instead.

adaptfilt Object	Replacement System Object
adaptfilt.lms adaptfilt.nlms adaptfilt.se adaptfilt.sd adaptfilt.ss	dsp.LMSFilter
adaptfilt.blms	dsp.BlockLMSFilter
adaptfilt.rls adaptfilt.qrdrls adaptfilt.swrls adaptfilt.hrls adaptfilt.hswrls	dsp.RLSFilter
adaptfilt.ftf adaptfilt.swftf	dsp.FastTransversalFilter
adaptfilt.ap adaptfilt.apru adaptfilt.bap	dsp.AffineProjectionFilter
adaptfilt.gal adaptfilt.lsl adaptfilt.qrdlsl	dsp.AdaptiveLatticeFilter
adaptfilt.filtxlms	dsp.FilteredXLMSFilter
adaptfilt.fdaf adaptfilt.ufdaf	dsp.FrequencyDomainAdaptiveFilter

adaptfilt Object	Replacement System Object
adaptfilt.blmsfft	dsp.LMSFilter, dsp.BlockLMSFilter, dsp.RLSFilter, dsp.FastTransversalFilter, dsp.AffineProjectionFilter, dsp.AdaptiveLatticeFilter, dsp.FilteredXLMSFilter, dsp.FrequencyDomainAdaptiveFilter
adaptfilt.adjllms adaptfilt.dllms	dsp.LMSFilter, dsp.BlockLMSFilter, dsp.RLSFilter, dsp.FastTransversalFilter, dsp.AffineProjectionFilter, dsp.AdaptiveLatticeFilter, dsp.FilteredXLMSFilter, dsp.FrequencyDomainAdaptiveFilter
adaptfilt.pbfdaf adaptfilt.pbufdaf	dsp.LMSFilter, dsp.BlockLMSFilter, dsp.RLSFilter, dsp.FastTransversalFilter, dsp.AffineProjectionFilter, dsp.AdaptiveLatticeFilter, dsp.FilteredXLMSFilter, dsp.FrequencyDomainAdaptiveFilter
adaptfilt.tdafdct adaptfilt.tfafdft	dsp.LMSFilter, dsp.BlockLMSFilter, dsp.RLSFilter, dsp.FastTransversalFilter, dsp.AffineProjectionFilter, dsp.AdaptiveLatticeFilter, dsp.FilteredXLMSFilter, dsp.FrequencyDomainAdaptiveFilter

Removal of `qfft` and `qformat` functions

The functions `qfft` and `qformat` have been removed. Use the `dsp.FFT` System object instead.

Removal of HDL Minimum Resource FFT block

The HDL Minimum Resource FFT block will be removed in a future release. In R2017b, the HDL Minimum Resource FFT block returns a warning. Use the `Burst Radix 2` architecture of the FFT HDL Optimized block instead. This block is in the `Transforms (dspxfm3)` library.

Removal of Streaming Radix 2 architecture in HDL-optimized FFT blocks and System objects

The `Streaming Radix 2` architecture has been removed from HDL-optimized FFT blocks and System objects for this release. Use the `Streaming Radix 22` architecture instead, which results in better hardware performance.

When you open a model containing an FFT HDL Optimized or IFFT HDL Optimized block that uses the `Streaming Radix 2` architecture, the block is automatically converted to use the `Streaming Radix 22` architecture. This change affects the latency of the block, so you must adjust the delay balancing of parallel data paths. The new latency is displayed on the block.

`dsp.HDLFFT` and `dsp.HDLIFFT` System objects that use the `Streaming Radix 2` architecture now return errors.

R2017a

Version: 9.4

New Features

Bug Fixes

Compatibility Considerations

Improved Spectrum Analyzer: Analyze signals in the frequency domain using additional units, dual visualization, and mask compliance output

Signal analysis with the Spectrum Analyzer block and System object has been improved:

- Visualize your signal spectrum as the root-mean squares (RMS) by using **Type > RMS**. When you select RMS as your spectrum type, you can choose from two spectrum units **Vrms** and **dBV**. Previously, these units were called power units.
- View the signal spectrum and spectrogram at the same time with the **View** setting. Use the **Axes layout** setting to set the dual view mode orientation: vertically stacked or side by side. You can also still view the spectrum and spectrogram individually.
- When you add a spectral mask, see when the spectrum values are inside or outside the limits. When the spectrum is inside the limits, the mask is green. When the spectrum is outside the limits, the mask is red.

Also, you can get statistics about what percentage of time the spectral mask passed or failed using the `getSpectralMaskStatus` function or the status bar tooltip.

- On the **Distortion Measurements** pane, measure up to 99 harmonics using the **Num. Harmonics** option.

For more information, see the `dsp.SpectrumAnalyzer` System object or the Spectrum Analyzer block.

Compatibility Considerations

Changed System Object Properties

Pre-R2017a Funtionality	Use Instead	Considerations
PowerUnits	SpectrumUnits	The property has been renamed. Update any existing code with the new property name.
SpectrumType = 'Spectrogram'	ViewType = 'Spectrogram'	The property value Spectrogram is now a view type. Update any existing code with the new property name.

Unified interface for `dsp.LogicAnalyzer`: Visualize, measure, and analyze signal transitions in MATLAB using the same interface as the Simulink Logic Analyzer

The `dsp.LogicAnalyzer` System object has improved runtime and interaction performance, memory usage, and a unified interface with the Simulink® **Logic Analyzer**.

Compatibility Considerations

The `DisplayChannelColor` property now supports customizable colors. String specifications continue to be supported and are converted to [R G B] values.

The default `DisplayChannelFormat` property is now 'auto'.

The `DisplayChannelHeight` and `DisplayChannelSpacing` properties are now defined in terms of pixels.

The `ReduceUpdates` and `MaxNumTimeSteps` properties are no longer used and will be removed in a future release.

Channelizer and Channel Synthesizer Blocks: Analyze and synthesize narrow subbands of a broadband signal using a polyphase FFT filter bank in Simulink

The Channelizer block implements an analysis filter bank that splits a broadband input signal into multiple narrowband signals. The Channel Synthesizer block merges multiple narrowband signals to form a single broadband signal. These filter banks are implemented using an FFT-based polyphase structure.

Asynchronous Buffering: Exchange signals at different rates and array sizes with the `dsp.AsyncBuffer` System object

You can write and read data from a FIFO buffer at different rates and frame sizes using the `dsp.AsyncBuffer` System object. The data that you write occupies the next available empty space in the buffer. After the last space in the buffer is used, the object

overwrites the oldest data. The buffer does not erase the data when the object reads it, so you can reread data from the past (overlap reading).

HDL Optimized Filters: Model and generate optimized hardware implementations for FIR filters and polyphase filter banks (requires HDL Coder for code generation)

Discrete FIR Filter

This Discrete FIR Filter HDL Optimized block and `dsp.HDLFIRFilter` System object model FIR filter structures optimized for HDL code generation with HDL Coder™. The filter is sample-based and includes control signals for flow control. Resource sharing options allow for tradeoffs between throughput and resource utilization. The block and object provide cycle-accurate models of the generated HDL code.

Polyphase Filter Bank

The Channelizer HDL Optimized block and `dsp.HDLChannelizer` System object model a polyphase filter bank and fast Fourier transform and support HDL code generation with HDL Coder. The algorithm provides an efficient hardware implementation and hardware-friendly control signals. You can achieve giga-sample-per-second (GSPS) throughput using vector input.

Frame Input Support for FIR Decimation

You can now generate HDL code from the FIR Decimation block when using frame input. The block accepts a column vector of input data, where each element of the vector represents a sample in time. The coder implements a parallel HDL architecture for the filter. This capability increases throughput in hardware designs. To configure the block for frame input:

- 1 Connect a column vector signal to the FIR Decimation block input port.
- 2 Set **Input processing** to `Columns as channels` (frame based).
- 3 Set **Rate options** to `Enforce single-rate processing`.
- 4 Right-click the block and select **HDL Code > HDL Block Properties**. Set the **Architecture** to `Frame Based`. The block implements a parallel HDL architecture. See [Frame-Based Architecture](#).

To generate HDL code, you must have an HDL Coder license. For information on HDL support for this block, see FIR Decimation.

Remove outliers from streaming signals in MATLAB and Simulink using Hampel filter

The `dsp.HampelFilter` System object in MATLAB and the Hampel Filter block in Simulink remove outliers from streaming signals by using the Hampel identifier.

Spectral estimation using filter bank in Simulink

To estimate the spectrum of a streaming signal in Simulink by using an analysis filter bank, set the **Method** parameter of the Spectrum Estimator block to `Filter bank`. For a given signal length, the filter bank approach of spectral estimation provides lower spectral leakage, higher frequency resolution, and a more accurate noise floor compared to the Welch's method.

Tunable UDP port number in generated code

The following System object properties and block parameters are now tunable in C/C++ generated code:

- `LocalIPPort` property in `dsp.UDPReceiver` System object.
- `RemoteIPPort` property in `dsp.UDPSender` System object.
- **Local IP port** parameter in UDP Receive block.
- **Remote IP port** parameter in UDP Send block.

These properties and parameters are not tunable during simulation. They are tunable only when you execute the generated code.

Filter signals using the `dsp.FilterCascade` System object

You can filter streaming signals with a cascade of filters using the `step` method of the `dsp.FilterCascade` System object. For a list of System objects that can be used as stages of the filter cascade, at the MATLAB command prompt, enter `dsp.FilterCascade.helpSupportedSystemObjects`.

Use delay and scalar gain in dsp.FilterCascade System object

You can now use the `dsp.Delay` System object and a numeric scalar value as stages in the `dsp.FilterCascade` System object.

Cascade a dsp.FilterCascade System object

The `dsp.FilterCascade` System object can now cascade another `dsp.FilterCascade` System object.

For example:

```
cicdecim = dsp.CICDecimator('DecimationFactor', 6, ...
                           'NumSections', 6);
decimcasc = dsp.FilterCascade(cicdecim, 1/gain(cicdecim));
ciccomp = dsp.CICCompensationDecimator;
filtchain = dsp.FilterCascade(decimcasc, ciccomp)
```

```
filtchain =
```

```
    dsp.FilterCascade with properties:
```

```
    Stage1: [1×1 dsp.FilterCascade]
    Stage2: [1×1 dsp.CICCompensationDecimator]
```

Access the complete history of LMS filter weights in MATLAB

When you set the `WeightsOutput` property of the `dsp.LMSFilter` System object to `'All'`, the object outputs a *FrameLength*-by-*Length* matrix of weights. *FrameLength* is the frame size of the input. *Length* is the length of the LMS filter. This matrix of weights corresponds to the full sample-by-sample history of weights values for all *FrameLength* samples of the input values.

Tab Completion: Complete parameter names and options in DSP System Toolbox System objects

When creating DSP System Toolbox System objects, you can use the Tab key to complete parameter names and options. For example, if you type `dsp.FIRFilter('` and press **Tab**, MATLAB displays a list of possible parameter names and options for the `dsp.FIRFilter` object.

Completion of parameters and options is not available for DSP System Toolbox functions.

Filter Builder and fdesign support IIR halfband filter System objects

The **Filter Builder** app and the `fdesign` function now support the `dsp.IIRHalfbandDecimator` and `dsp.IIRHalfbandInterpolator` System objects.

For example:

- **Filter Builder** — When you choose the filter response as **Halfband**, select the **Impulse response** as IIR, and set **Filter Type** to either Decimator or Interpolator, the filterBuilder designs a `dsp.IIRHalfbandDecimator` and `dsp.IIRHalfbandInterpolator` System objects, respectively.
- `fdesign` — 'SystemObject' flag set to true is supported in the design method of `fdesign.decimator`, `fdesign.interpolator`, `fdesign.halfband`, and `fdesign.nyquist` objects while designing an IIR halfband decimator or interpolator filter.

```
Fs = 2000; % Sampling frequency of input signal
M = 2; % Decimation factor
TW = 100; % Transition width of filter to be designed, 100 Hz
Ast = 60; % Stopband attenuation of filter to be designed, 80 db
% Store decimator design specs
f = fdesign.decimator(2, 'halfband', 'TW,Ast', TW, Ast, Fs);
iirhalfbanddecim = design(f, 'iirlinphase', 'FilterStructure', 'iirdecim', ...
    'SystemObject', true)
```

```
iirhalfbanddecim =
```

```
dsp.IIRHalfbandDecimator with properties:
```

```
Specification: 'Coefficients'
Structure: 'Minimum multiplier'
HasPureDelayBranch: true
Delay: 14
AllpassCoefficients2: [7x2 double]
HasTrailingFirstOrderSection: false
```

Specify image file icons for MATLAB System block

You can specify a MATLAB System block icon as an image file using a new option in the MATLAB Editor. While editing your System object, specify the image file by selecting **System Block > Add Image Icon**. After specifying the image file, this code is added to the System object class:

```
function icon = getIconImpl(~)
    % Define icon for System block
    icon = matlab.system.display.Icon('image.png');
end
```

For more information, see [Customize System Block Appearance](#).

Change tunable System object properties before locking

For independent tunable properties, you can now change the value at any time.

If the tunable property has a dependent datatype property, you must lock the object before changing the property.

Support for Time Scope to For Each subsystems

You can place a Time Scope block within a For Each subsystem block. The scope follows the same behavior as the Display block. For example, within a For Each subsystem, the Time Scope block displays only the last iteration of the For Each subsystem.

Copy scope to clipboard

To share the output of a signal simulation, copy the scope graphic to your clipboard by selecting **File > Copy to Clipboard**. The scope colors are converted to a print-friendly coloring. See [Share Scope Image](#).

Interactive legend for scopes

If you show a legend on your scope block or System object, you can use the legend to filter which signals are shown. Left-clicking a signal in the legend hides all other signals in the scope. Right-clicking a signal in the legend toggles whether the scope shows or hides the signal.

Stem plot option for Time Scope block

In the Time Scope block, you can visualize your signal as a stem plot. From the **View > Style** menu, select **Plot type > Stem**.

Time Scope Block: Connect nonvirtual bus and array of buses signals

You can connect nonvirtual bus signals and array of buses signals to a Time Scope block. To display the signals in the Time Scope block, use the normal or accelerator simulation mode. For details, see Nonvirtual Bus and Array of Buses Signals and Save Simulation Data Using a Scope Block.

Frame-based processing changes

As part of the changes in how DSP System Toolbox handles frame-based processing, certain block options have been removed and certain function options error. The following sections provide more detailed information about the specific R2017a DSP System Toolbox software changes for frame-based processing:

- “Input processing parameter set to Inherited” on page 2-9
- “InputProcessing property set to Inherited errors” on page 2-11
- “Rate options parameter set to Inherit from input” on page 2-11
- “Find the histogram over parameter set to Inherited” on page 2-11
- “Running difference parameter set to Inherit from input” on page 2-12
- “Save 2-D signals as parameter set to Inherit from input” on page 2-12
- “Treat Mx1 and unoriented sample-based signals as parameter removed” on page 2-12
- “Sample-based processing parameter removed” on page 2-13

Input processing parameter set to Inherited

The `Inherited` option has been removed from the **Input processing** parameter of the following blocks:

- Biquad Filter
- Arbitrary Response Filter — To see the change, select **Use basic elements to enable filter customization**.
- Bandpass Filter — To see the change, select **Use basic elements to enable filter customization**.
- Bandstop Filter — To see the change, select **Use basic elements to enable filter customization**.

- CIC Filter — To see the change, select **Use basic elements to enable filter customization**.
- Comb Filter — To see the change, select **Use basic elements to enable filter customization**.
- Hilbert Filter — To see the change, select **Use basic elements to enable filter customization**.
- Inverse Sinc Filter — To see the change, select **Use basic elements to enable filter customization**.
- Nyquist Filter — To see the change, select **Use basic elements to enable filter customization**.
- Octave Filter — To see the change, select **Use basic elements to enable filter customization**.
- Digital Filter Design — To see the change, select **Use basic elements to enable filter customization**.
- CIC Interpolation
- Downsample
- Upsample
- Repeat
- Variable Fractional Delay
- Mean — To see the change, select **Running mean**.
- Variance — To see the change, select **Running variance**.
- RMS — To see the change, select **Running RMS**.
- Standard Deviation — To see the change, select **Running standard deviation**.
- Minimum — To see the change, set **Mode** to *Running*.
- Maximum — To see the change, set **Mode** to *Running*.
- Cumulative Sum
- Cumulative Product
- Edge Detector
- Zero Crossing
- Unwrap

Compatibility Considerations

Blocks using the `Inherited` option in models created in previous releases can no longer use this option in R2017a. When you open these models in R2017a, the block chooses `Columns as channels` (frame based).

InputProcessing property set to Inherited errors

Setting `Input processing` property to `Inherited` now causes an error in these functions:

- `block`
- `realizemdl`

It is recommended that you set this property to one these options:

- `'columnsaschannels'` — Treat each column of the input signal as an independent channel.
- `'elementsaschannels'` — Treat each element of the input signal as an independent channel.

Rate options parameter set to Inherit from input

The `Inherit from input` option has been removed from the **Rate options** parameter of the CIC Decimation block when the input to the block is a scalar.

Compatibility Considerations

If the **Rate options** parameter in the existing model is set to `Inherit from input`, the model errors in R2017a. It is recommended that you update this parameter to either `Allow multirate processing` or `Enforce single-rate processing`.

For information on which **Rate options** parameter to choose, see *Rate options parameter set to Inherit from input* under *Frame-based processing* in the R2015a DSP System Toolbox release notes.

Find the histogram over parameter set to Inherited

The `Inherited` option has been removed from the **Find the histogram over** parameter of the Histogram block.

Compatibility Considerations

In existing models, Histogram blocks using the `Inherited` option can no longer use this option in R2017a. When you open these models in R2017a, the block chooses `Each column`.

Running difference parameter set to Inherit from input

The `Inherit from input` option has been removed from the **Running difference** parameter of the Difference block.

Compatibility Considerations

In existing models, Difference blocks using the `Inherit from input` option can no longer use this option in R2017a. When you open these models in R2017a, the block chooses `No`.

Save 2-D signals as parameter set to Inherit from input

The `Inherit from input` option has been removed from the **Save 2-D signals as** parameter of the Triggered To Workspace block.

Compatibility Considerations

Triggered To Workspace blocks using the `Inherit from input` option in existing models can no longer use this option in R2017a. When you open these models in R2017a, the block chooses `2-D array (concatenate along first dimension)`.

Treat Mx1 and unoriented sample-based signals as parameter removed

The **Treat Mx1 and unoriented sample-based signals as** parameter has been removed from these blocks:

- Buffer
- Delay Line

Compatibility Considerations

In existing models, Buffer and Delay Line blocks using the **Treat Mx1 and unoriented sample-based signals as** parameter can no longer use this parameter in R2017a. The blocks treat these inputs as single channels.

Sample-based processing parameter removed

The **Sample-based processing** parameter has been removed from the Unbuffer block.

Compatibility Considerations

In existing models, Unbuffer blocks using the **Sample-based processing** parameter can no longer use this parameter in R2017a. The block automatically unbuffers an M -by- N matrix input into a 1-by- N output vector.

Functionality being removed

- “Running Mode in Statistics Objects and Blocks” on page 2-13
- “Audio device recorder and player objects” on page 2-14
- “Radix 2 architecture of HDL-optimized FFT blocks and System objects” on page 2-15

Running Mode in Statistics Objects and Blocks

The running mode in the following System objects and blocks will be removed in a future release.

System Object	Use This Instead
dsp.Maximum	dsp.MovingMaximum
dsp.Minimum	dsp.MovingMinimum
dsp.Mean	dsp.MovingAverage
dsp.RMS	dsp.MovingRMS
dsp.StandardDeviation	dsp.MovingStandardDeviation
dsp.Variance	dsp.MovingVariance

Blocks	Use This Instead
Maximum	Moving Maximum
Minimum	Moving Minimum
Mean	Moving Average
RMS	Moving RMS
Standard Deviation	Moving Standard Deviation
Variance	Moving Variance

Audio device recorder and player objects

The following DSP features warn in R2017a and will be removed in a future release.

System object	Use This Instead
<code>dsp.AudioRecorder</code>	<p><code>audioDeviceReader</code> object in Audio System Toolbox™.</p> <hr/> <p>Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box has been removed. You can specify the driver in the <code>audioDeviceReader</code> object by using the <code>Driver</code> property.</p>
<code>dsp.AudioPlayer</code>	<p><code>audioDeviceWriter</code> object.</p> <hr/> <p>Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box is removed. You can specify the driver in the <code>audioDeviceWriter</code> object by using the <code>Driver</code> property.</p>

Compatibility Considerations

Existing instances of these System objects continue to run. For new instances of the functionality, use the replacement feature.

Radix 2 architecture of HDL-optimized FFT blocks and System objects

The `Radix 2` architecture, used in the FFT HDL Optimized and IFFT HDL Optimized blocks, and in the `dsp.HDLFFT` and `dsp.HDLIFFT` System objects, will be removed in a future release. Use of this architecture returns a warning in R2017a. Use the `Radix 2^2` architecture instead, which results in better hardware performance.

Architecture	Use This Instead
Streaming Radix 2	Streaming Radix 2^2

Compatibility Considerations

The latency of the two architectures is different. The new latency is displayed on the blocks, or can be accessed using the `getLatency` method of the System object. When you change the architecture, adjust any delay balanced paths in your model.

R2016b

Version: 9.3

New Features

Bug Fixes

Compatibility Considerations

Logic Analyzer: Visualize, measure, and analyze transitions and states over time for Simulink signals

The Logic Analyzer visualization tool enables you to view the transitions of signals. You can use the **Logic Analyzer** to:

- Debug and analyze models.
- Trace and correlate many signals simultaneously.
- Detect and analyze timing violations.
- Trace system execution.

See [Inspect and Measure Transitions Using the Logic Analyzer](#) to explore some of its key functionality.

Spectral Mask: Compare a signal spectrum to a spectral mask using Spectrum Analyzer

The Spectrum Analyzer block and `SpectrumAnalyzer` System object support defining and overlaying a spectral mask on spectrum plots. Spectral masks are useful for verifying that the signal spectrum is within an area of defined spectral limitations.

Channelizer and Channel Synthesizer: Analyze and synthesize narrow subbands of a broadband signal using a polyphase FFT filter bank

The `dsp.Channelizer` System object implements an analysis filter bank that splits a broadband input signal into multiple narrowband signals. The `dsp.ChannelSynthesizer` System object merges multiple narrowband signals to form a single broadband signal. These filter banks are implemented using an FFT based polyphase structure.

Moving Statistics: Measure descriptive statistics on streaming signals in MATLAB and Simulink

Compute moving statistics such as the average, RMS, standard deviation, variance, minimum, maximum, and median of a streaming input signal in MATLAB code and Simulink models.

Gigasample per Second (GSPS) Signal Processing: Increase the throughput of HDL code generated from Discrete FIR Filter and Integer Delay blocks using frame input

You can now generate HDL code from the Discrete FIR Filter block when using frame input. First set **Input processing** to `Columns as channels (frame based)`. Then right-click the block, open **HDL Code > HDL Block Properties**, and set the **Architecture** to `Frame Based`. The block accepts vector input data, where each element of the vector represents a sample in time. The coder implements a parallel HDL architecture for the filter. For information on HDL support for this block, see `Discrete FIR Filter`.

The Delay block also supports HDL code generation with frame input data. Set **Input processing** to `Columns as channels (frame based)`. The block accepts vector input data, where each element of the vector represents a sample in time.

This capability increases throughput in hardware designs. To generate HDL code, you must have an HDL Coder license.

Stream signals to and from binary files

You can now read and write binary files in the MATLAB environment using the `dsp.BinaryFileReader` and `dsp.BinaryFileWriter` System objects. In the Simulink environment, you can use the corresponding Binary File Reader and Binary File Writer blocks. The reader can read any binary file and does not depend on how a binary file is created.

Compute LMS adaptive filter weights using LMS Update block

LMS Update block estimates adaptive filter weights using an LMS adaptive algorithm. The block accepts the data and error as inputs and computes the filter weights based on the specified LMS algorithm. Using this block, you can model and simulate variants of LMS adaptive filtering algorithm, including the filtered-X LMS.

Allpass Filter block

The Allpass Filter block filters each channel of a streaming input signal using a single-section or a multiple-section allpass filter in Simulink. You can implement the allpass filter in minimum multiplier, wave digital filter, or lattice form.

Specify coefficients in Farrow Rate Converter block and System object

You can now specify the coefficients of Farrow rate converters directly. To specify the coefficients, set the `Specification` property of the `dsp.FarrowRateConverter` System object to `Coefficients`. In the Farrow Rate Converter block, set the **Specification method** parameter to `Coefficients`.

Spectral estimation using filter banks

To estimate the spectrum of a signal using an analysis filter bank, set the `Method` property of the `dsp.SpectrumEstimator` System object to `'Filter bank'`. The filter bank approach provides low spectral leakage, high frequency resolution, and an accurate noise floor.

High-throughput polyphase filter bank for HDL example

The Generate HDL Code for High Throughput Signal Processing model example shows how to design a polyphase filter bank to achieve gigasample per second data rates in the generated HDL implementation. The model uses the FFT HDL Optimized block with vector input.

Bit-reversed input order for HDL-optimized FFT

For vector input data, the HDL-optimized FFT algorithms now support bit-reversed input with natural order output. For scalar input data, you can select any input order with any output order. The default is natural order input with bit-reversed output.

This change affects these blocks and System objects:

- FFT HDL Optimized
- IFFT HDL Optimized

-
- `dsp.HDLFFT`
 - `dsp.HDLIFFT`

HDL code generation for reset port on Discrete FIR Filter

You can now generate HDL code from the Discrete FIR Filter block when you configure the block to have an external reset port.

Compiler support for System object scopes

You can now compile MATLAB code containing calls to `dsp.ArrayPlot`, `dsp.SpectrumAnalyzer`, or `dsp.TimeScope` System objects by using the `mcc` MATLAB compiler command. You use compiled MATLAB code to create a standalone application.

Custom X-axis data in Array Plot

You can now customize X-axis data in the Array Plot block and `dsp.ArrayPlot` System object. Use this option to specify the axis for arbitrarily spaced data.

Set legend strings and autoscaling programmatically in Time Scope

In the `dsp.TimeScope` System object, you can control the appearance of the scope from the MATLAB command line or from within MATLAB code. Use the `ChannelNames` property to specify a cell array of names to use in the plot legend. When `ShowLegend` is `true`, the legend pulls the names from the cell array. Use the `AxesScaling` property to enable autoscaling of the plot. The default scaling setting is to scale the plot when the simulation stops.

Simpler way to call System objects

Instead of using the `step` method to perform the operation defined by a System object, you can call the object with arguments, as if it were a function. The `step` method continues to work. This feature improves the readability of scripts and functions that use many different System objects.

For example, if you create a `dsp.FFT` System object named `fft1024`, then you call the System object as a function with that name.

```
fft1024 = dsp.FFT('FFTLengthSource','Property', ...  
    'FFTLength',1024);  
fft1024(x)
```

The equivalent operation using the `step` method is:

```
fft1024 = dsp.FFT('FFTLengthSource','Property', ...  
    'FFTLength',1024);  
step(fft1024,x)
```

When the `step` method has the System object as its only argument, the function equivalent has no arguments. You must call this function with empty parentheses. For example, `step(sysobj)` and `sysobj()` perform equivalent operations.

System objects support for additional inputs, global variables, and enumeration data types

- System objects in code generated using MATLAB Coder can have up to 1024 inputs.
- You can use global variables declared in System objects to exchange data with the Data Store Memory block in Simulink. You can use these variables in generated code.
- Enumeration data types for System objects included in Simulink using the MATLAB System block is supported. Enumerations restrict data to a finite set of data values that inherit from `int8`, `uint8`, `int16`, `uint16`, `int32`, or `Simulink.IntEnumType` data types, or a data type you define using `Simulink.defineIntEnumType`.

Functionality being removed

Removal of sample mode from the DSP System Toolbox System objects

The `FrameBasedProcessing` property of all DSP System objects will be removed in a future release. System objects containing this property will then work only in frame-based processing mode. See [What Is Frame-Based Processing?](#) for more information. If this property is set to `true`, no further change to the input is required. If this property is set to `false` and the input is a column vector or an N -D matrix, reshape the input such that each column in the input is an independent channel.

Effective R2016b, modifying this property throws an error for these System objects:

- `dsp.AllpoleFilter`

-
- `dsp.AnalyticSignal`
 - `dsp.BiquadFilter`
 - `dsp.Buffer`
 - `dsp.CumulativeProduct`
 - `dsp.CumulativeSum`
 - `dsp.Delay`
 - `dsp.FIRFilter`
 - `dsp.IIRFilter`
 - `dsp.MatFileReader`
 - `dsp.MatFileWriter`
 - `dsp.Maximum`
 - `dsp.Mean`
 - `dsp.Minimum`
 - `dsp.PeakToPeak`
 - `dsp.PeakToRMS`
 - `dsp.PhaseUnwrapper`
 - `dsp.RMS`
 - `dsp.SignalSink`
 - `dsp.StandardDeviation`
 - `dsp.VariableFractionalDelay`
 - `dsp.VariableIntegerDelay`
 - `dsp.Variance`
 - `dsp.ZeroCrossingDetector`

Digital Filter block and System object

The `dsp.DigitalFilter` System object has been removed and the Digital Filter block will be removed in a future release. The table lists the recommended replacement blocks and System objects.

Filter to Implement	Replacement Block	Replacement System Object
FIR filter structures	Discrete FIR Filter	<code>dsp.FIRFilter</code>
Biquad (SOS) structures	Biquad Filter	<code>dsp.BiquadFilter</code>

Filter to Implement	Replacement Block	Replacement System Object
Non-SOS IIR filter structures	Discrete Filter	<code>dsp.IIRFilter</code>
Allpole structures	Allpole Filter	<code>dsp.AllpoleFilter</code>

Compatibility Considerations

Existing instances of the Digital Filter block continue to run. For new models, use the replacement block listed in the table.

Removal of `adaptfilt` objects

Starting in R2016b, using `adaptfilt` objects throws an error. In a future release, these objects will be removed. Use the corresponding System object instead.

<code>adaptfilt</code> Object	Replacement System Object
<code>adaptfilt.lms</code> <code>adaptfilt.nlms</code> <code>adaptfilt.se</code> <code>adaptfilt.sd</code> <code>adaptfilt.ss</code>	<code>dsp.LMSFilter</code>
<code>adaptfilt.blms</code>	<code>dsp.BlockLMSFilter</code>
<code>adaptfilt.rls</code> <code>adaptfilt.qrdrls</code> <code>adaptfilt.swrls</code> <code>adaptfilt.hrls</code> <code>adaptfilt.hswrls</code>	<code>dsp.RLSFilter</code>
<code>adaptfilt.ftf</code> <code>adaptfilt.swftf</code>	<code>dsp.FastTransversalFilter</code>

adaptfilt Object	Replacement System Object
adaptfilt.ap adaptfilt.apru adaptfilt.bap	dsp.AffineProjectionFilter
adaptfilt.gal adaptfilt.lsl adaptfilt.qrdsl	dsp.AdaptiveLatticeFilter
adaptfilt.filtxlms	dsp.FilteredXLMSFilter
adaptfilt.fdaf adaptfilt.ufdaf	dsp.FrequencyDomainAdaptiveFilter
adaptfilt.blmsfft	Will be removed in a future release
adaptfilt.adjlm adaptfilt.dlms	Will be removed in a future release
adaptfilt.pbfdaf adaptfilt.pbufdaf	Will be removed in a future release
adaptfilt.tdafdct adaptfilt.tfafdft	Will be removed in a future release

Cell array support removal for dsp.AllpassFilter coefficients

Cell array support for the `LatticeCoefficients`, `AllpassCoefficients`, and `WDFCoefficients` properties of `dsp.AllpassFilter` System object will be removed in a future release. Use an N -by-1 or N -by-2 numeric array instead.

Inherited option removed from the input processing parameter

The `Inherited` option has been removed from the **Input processing** parameter in the Analytic Signal block.

Compatibility Considerations

In models created before R2016b that have **Input processing** set to `Inherited`, the parameter changes to `Columns as channels (frame based)` if you open the model in R2016b or after. If the block input is sample-based, nonscalar, and not a column vector, change **Input processing** to `Elements as channels (sample based)`.

Frame status parameter removed from the Check Signal Attributes block

The **Frame status** parameter has been removed from the Check Signal Attributes block.

qfft object errors

Starting in R2016b, using the `qfft` object throws an error. Use the `dsp.FFT System` object instead.

dspstartup removed

`dspstartup` has been removed. Use the DSP Simulink Model Templates instead.

R2016a

Version: 9.2

New Features

Bug Fixes

Compatibility Considerations

DSP Unfolding for Mac: Generate multithreaded MEX files from MATLAB functions on Mac OS X

DSP unfolding is a technique to improve throughput through parallelization. In R2016a, `dspunfold` function implements DSP unfolding on Mac OS X platform as well as Windows and Linux. The multithreaded MEX file that `dspunfold` generates from the specified MATLAB function leverages the multicore CPU architecture of the host computer, and can improve speed significantly.

Faster FIR and Biquad Filters: Run faster simulations for system models that include FIR and biquad filters

These filters have significantly faster simulation speeds for multichannel inputs:

- `dsp.FIRFilter` System object and Discrete FIR Filter block in the `Direct form` structure
- `dsp.BiquadFilter` System object and Biquad Filter block in the `Direct form II transposed` structure

Fixed-Point Farrow Rate Converter: Design and simulate Farrow rate conversion filters using fixed-point data types

The `dsp.FarrowRateConverter` System object and Farrow Rate Converter block now support fixed-point data types.

Gigasample per Second (GSPS) Signal Processing: Increase throughput of HDL-optimized FFT and IFFT algorithms using frame input

You can increase the throughput of the FFT and IFFT calculation by using vector input and output ports. The internal algorithm computes the FFT or IFFT of each vector element in parallel.

The FFT implementation is now a Radix 2^2 architecture which improves performance for vector input. The table compares hardware implementation resources between the old Radix 2 architecture and the new Radix 2^2 architecture.

Architecture	Multipliers	Adders	Memory	Control Logic For Vector Input
Radix 2 Hybrid	$\log_4(N-1)$	$3 \times \log_4(N)$	$17N/16 - 1$	Complicated
Radix 2^2 (SDF)	$\log_4(N-1)$	$4 \times \log_4(N)$	$N - 1$	Simple

This change affects these blocks and System objects:

- FFT HDL Optimized
- IFFT HDL Optimized
- `dsp.HDLFFT`
- `dsp.HDLIFFT`

HDL Optimizations for Biquad Filter: Reduce critical path or area when generating HDL from a subsystem that includes a Biquad Filter block

The Biquad Filter block is now included in subsystem optimizations for speed and area of the generated HDL. To specify resource sharing, streaming, and pipeline options, right-click the subsystem containing the Biquad Filter block and open the **HDL Code > HDL Properties** dialog box. To use these optimizations you must set the **Architecture** of the Biquad Filter block to `Fully parallel`. This feature requires an HDL Coder license.

The optimizations work the same way as the optimizations for the Discrete FIR Filter block. You can share resources between Biquad Filter and Discrete FIR Filter blocks in the same subsystem. See Subsystem Optimizations for Filters.

Differentiate a signal using the `dsp.Differentiator` System object and Differentiator block

`dsp.Differentiator` System object applies a direct form FIR full band differentiator filter on an input signal. The object uses an FIR equiripple filter design to design the differentiator filter. The Differentiator Filter block imports the functionality of this object into the Simulink environment.

Play audio data using the `audioDeviceWriter` System object and Audio Device Writer block

Play audio data on your computer's audio device using the `audioDeviceWriter` System object and Audio Device Writer block. Audio System Toolbox provides enhanced

functionality. For instance, you can customize the channel-to-speaker mapping and monitor the dropouts in the audio data. Audio System Toolbox also adds support for low-latency ASIOo drivers on Windows.

Specify coefficients in IIR Halfband Interpolator and IIR Halfband Decimator Blocks and System objects

You can now specify the coefficients of IIR halfband filters directly. To specify the coefficients, set the `Specification` property of the `dsp.IIRHalfbandInterpolator` and `dsp.IIRHalfbandDecimator` System objects to `Coefficients`. In the blocks, set the **Filter specification** parameter to `Coefficients`.

In this mode:

- You can specify the structure as `Minimum multiplier` or `Wave Digital Filter`. In both forms, you specify the coefficients through the applicable properties.
- The coefficients in the `Minimum multiplier` form are tunable, that is, you can change them even after the object is locked or during the model simulation.
- The first branch of the polyphase filter structure can be modeled as a pure delay.
- The last section of the second branch of the polyphase filter structure can contain a first-order section.

Customize the data limits of the Matrix Viewer block

You can now customize the x-axis and y-axis limits of the Matrix Viewer block. Changing the limits on the block dialog box updates the block axes in real time.

Code generation for wave digital filter structure in `dsp.AllpassFilter` System object

With the `Structure` property of the `dsp.AllpassFilter` System object set to `Wave Digital Filter`, you can:

- Generate C code from the System object.
- Import this System object into Simulink using the MATLAB System block.
- Specify the last section of the filter as first order by setting the `TrailingFirstOrderSection` property to `true`.

-
- Assign array data to the `WDFCoefficients` property.

Compatibility Considerations

Old MATLAB code with `WDFCoefficients` property set to a cell array continues to run in R2016a. However, it is recommended that you set this property to an array value.

Generate coefficients for multirate filters

Design an FIR interpolator, decimator, and rate converter using the `designMultirateFIR` function. To compute the coefficients of the multirate filter, this function uses an FIR Nyquist filter. Coefficients are computed based on the rate conversion factor provided as the input to the function.

Select the color of the noise in `dsp.ColoredNoise` System object

In the `dsp.ColoredNoise` System object, you can now specify the color of the noise to generate by setting the `Color` property to:

- White
- Pink
- Brown
- Blue
- Purple
- Custom

When you choose `Custom`, you can specify the power density of the noise through the `InverseFrequencyPower` property.

Full-precision setting for product data type of Biquad Filter

You can now specify biquad filters to use full-precision rules for the product data type. In the `dsp.BiquadFilter` System object, set the `NumeratorProductDataType` and `DenominatorProductDataType` properties can be set to `Full` precision. In the Biquad Filter block, set the **Product output** parameter to `Inherit` via internal rule.

In the full-precision setting, the filter computes all internal arithmetic and output data types using full-precision rules. These rules provide more accurate fixed-point numerics that prevent quantization from occurring within the filter. Bits are added as needed so that no roundoff or overflow occurs.

Code generation for Subband Analysis and Subband Synthesis Filters

The `dsp.SubbandAnalysisFilter` and `dsp.SubbandSynthesisFilter` System objects can generate C code. When these objects run in single precision mode, you can also generate C code optimized for ARM® Cortex®-A and ARM Cortex-M processors. To generate code on ARM Cortex processors, you must have an Embedded Coder® license.

Enhancements to Variable Fractional Delay

- The Variable Fractional Delay block and `dsp.VariableFractionalDelay` System object accept variable-size input signals. During the block simulation or when the object is locked, you can vary the number of samples per channel of the input signal. However, you cannot vary the number of channels.
- Interpolation precision has improved. To improve the precision, set the Interpolation method is set to FIR.

Multiple inputs for Spectrum Analyzer

The Spectrum Analyzer block and `dsp.SpectrumAnalyzer` System object accept more than one input port. In the Spectrum Analyzer block, set the number of inputs by selecting **File > Number of Input Ports**. For `dsp.SpectrumAnalyzer`, set the `NumInputPorts` property to the number of ports you want. For the block and System object, all inputs must have the same frame size. For the block only, all inputs must also have the same sample time.

Additional axes for Time Scope

The Time Scope block and `dsp.TimeScope` System object now default to 4 x 4 separate axes and allow up to 16 x 16 axes. To set the axes select **View > Layout**. Then, drag the axes grid to the desired number of axes and positions of those axes.

Set legend programmatically in Array Plot

In the Array Plot block and `dsp.ArrayPlot` System object, you can control the channel names from the MATLAB command line or from within MATLAB code. Define the channel names in a cell array. The plot legend pulls the names from this cell array. For the block, set the names in the `ChannelNames` parameter. For the System object, set the names in the `ChannelNames` property.

System object property display

How System objects properties are displayed at the MATLAB command line has changed.

- Fixed-point properties are displayed only if you click a link at the end of a System object property display.
- Properties are grouped as defined in the `getPropertyGroupsImpl` method.
- If `getPropertyGroupsImpl` defines multiple sections, only properties from the first section group are displayed. To display additional properties, click the link at the end of a System object property display. Section groups are defined using `matlab.system.display.SectionGroup`. Group titles are also displayed. To omit the “Main” title for the first group of properties, in the `matlab.system.display.SectionGroup` class, set `TitleSource` to 'Auto'.

Compatibility Considerations

The `matlab.system.showFixedPointProperties` and `matlab.system.hideFixedPointProperties` functions have been removed. These functions controlled the display of fixed-point properties. If your code uses either of these functions, such as in a startup script, you now receive a warning. The **System Objects Preferences** panel in the MATLAB Preferences dialog box has also been removed. This panel was another way to set the fixed-point properties display.

System object enhancements to MATLAB System block

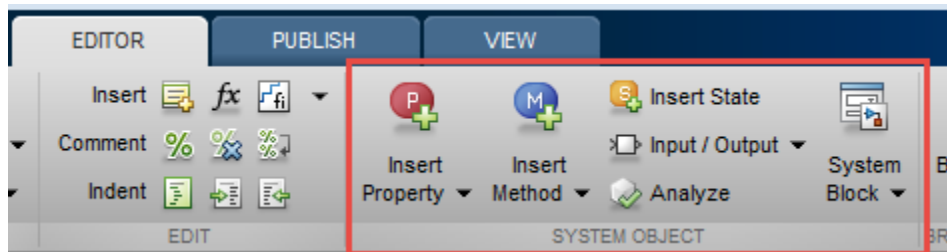
To implement these classes and methods for defining your own System objects, add them to your object's class definition file.

- Fixed-point data tab — The `showFiSettingsImpl` method adds a Data Types tab to the MATLAB System block dialog box. This tab includes options for fixed-point data settings.
- Model reference discrete sample time inheritance — The `allowModelReferenceDiscreteSampleTimeInheritanceImpl` method lets you specify whether a System object in a referenced model can inherit the sample time of the parent model. If your object uses discrete sample time in its algorithm, you set this method to `true` to allow inheritance.

Enhanced System Object Development with MATLAB Editor

Create System objects in the MATLAB Editor using code insertion and visualization options.

- Define your System object with options to insert properties, methods, states, inputs, and outputs.
- View and navigate the System object code with the Analyzer.
- Develop System block and preview block dialog box interactively (with Simulink only).



These coding tools are available when you open an existing System object or create a new System object with **New > System object**.

Functionality being removed

The following DSP features will be removed in a future release

Functions	Use This Instead
<code>fdesign.parmeq</code>	<code>fdesign.parmeq</code> function in Audio System Toolbox.

Functions	Use This Instead
<code>fdesign.octave</code>	<code>fdesign.octave</code> function in Audio System Toolbox.
<code>fdesign.audioweighting</code>	<code>fdesign.audioweighting</code> function in Audio System Toolbox.
<code>iirparameq</code>	<code>designParamEQ</code> function in Audio System Toolbox.
<code>midicallback</code>	<code>midicallback</code> function in Audio System Toolbox.
<code>midicontrols</code>	<code>midicontrols</code> function in Audio System Toolbox.
<code>midiid</code>	<code>midiid</code> function in Audio System Toolbox.
<code>midiread</code>	<code>midiread</code> function in Audio System Toolbox.
<code>midisync</code>	<code>midisync</code> function in Audio System Toolbox.
Blocks	Use This Instead
Audio Weighting Filter block	Audio Weighting Filter block in Audio System Toolbox.
Octave Filter block	Octave Filter block in Audio System Toolbox.
MIDI Controls block	MIDI Controls block in Audio System Toolbox.
Parametric EQ Filter block	Parametric EQ Filter block in Audio System Toolbox.

Blocks	Use This Instead
From Audio Device block	<p>Audio Device Reader block in Audio System Toolbox.</p> <hr/> <p>Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box is removed. You can specify the driver in Audio Device Reader block using the Driver parameter.</p>
To Audio Device block	<p>Audio Device Writer block.</p> <hr/> <p>Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box is removed. You can specify the driver in Audio Device Writer block using the Driver parameter.</p>
System object	Use This Instead
dsp.ParametricEQFilter	multibandParametricEQ object in Audio System Toolbox.
dsp.AudioRecorder	<p>audioDeviceReader object in Audio System Toolbox.</p> <hr/> <p>Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box is removed. You can specify the driver in audioDeviceReader object using the Driver property.</p>

System object	Use This Instead
dsp.AudioPlayer	audioDeviceWriter object . Note The ability to select or change the audio driver through the DSP System Toolbox preferences dialog box is removed. You can specify the driver in audioDeviceWriter object using the Driver property.
mfilt object	Use One of The Following
mfilt.linearinterp	<ul style="list-style-type: none"> • dsp.FarrowRateConverter object with PolynomialOrder property set to 1. This value achieves linear interpolation. • dsp.FIRInterpolator object with InterpolationFactor property set to 2. • dsp.CICInterpolator object with NumSections and InterpolationFactor properties set to 2.
qfft object	Use This Instead
qfft	dsp.FFT

Compatibility Considerations

Existing instances of these functions, System objects, and blocks continue to run. For new instances of the functionality, use the replacement feature.

R2015b

Version: 9.1

New Features

Bug Fixes

Compatibility Considerations

DSP Unfolding: Generate a multi-threaded MEX File from a MATLAB function

Generate a multi-threaded MEX file from a MATLAB function, using the `dspunfold` function. DSP unfolding is a technique to improve throughput through parallelization. The multi-threaded MEX file leverages the multicore CPU architecture of the host computer and can improve speed significantly.

HDL Optimizations for Discrete FIR Filter: Implement FIR filters in hardware at higher frequencies or using fewer resources

You can now optimize speed and area of the generated HDL for the Discrete FIR Filter block. Right-click the subsystem containing the Discrete FIR Filter block, and open the **HDL Code > HDL Properties** dialog to specify resource sharing, streaming, and pipeline options. You can use these optimizations when the **Architecture** is **Fully parallel**. This feature requires an HDL Coder license. See Discrete FIR Filter.

Array Plot Block: Visualize array and vector data

An Array Plot block for plotting arrays and vectors has been added to the `dspnks4` library.

Additional Multirate Filters: Design Halfband, CIC compensation, and HDL-optimized FIR rate conversion filters

Implement FIR and IIR halfband interpolator and decimator using these new design features:

- The `dsp.IIRHalfbandInterpolator` and `dsp.IIRHalfbandDecimator System` objects use efficient polyphase IIR halfband structure to interpolate and decimate an input signal, by a factor of two. Allpass filters, used in the polyphase branches, use elliptic or quasilinear phase design methods. You can use these System objects to implement the synthesis portion and the analysis portion of a two-band filter bank. For a Simulink implementation, use the IIR Halfband Interpolator and IIR Halfband Decimator blocks
- FIR Halfband Interpolator and FIR Halfband Decimator blocks use an FIR equiripple design to construct the halfband filter, they use an efficient polyphase

implementation to filter the input. These blocks implement the functionality of the `dsp.FIRHalfbandInterpolator` and `dsp.FIRHalfbandDecimator` System objects.

CIC Compensation Interpolator and CIC Compensation Decimator blocks compensate for the passband droop and wide transition region of CIC filters. These blocks implement the functionality of the `dsp.CICCompensationInterpolator` and `dsp.CICCompensationDecimator` System objects.

FIR Rate Conversion HDL Optimized block upsamples, filters, and downsamples a signal using an efficient polyphase FIR structure. The block operates on one sample at a time and provides hardware control signals to pace the flow of samples in and out of the block. The `dsp.HDLFIRRateConverter` System object provides equivalent MATLAB functionality. Both the block and System object support HDL code generation, when used with HDL Coder.

Conversion Filter Blocks: Convert the rate of signals in Simulink models

- The Farrow Rate Converter block uses a farrow structure to implement a polynomial-based filter that does the sample rate conversion. This converter can handle arbitrary rate change factors efficiently. This block implements the functionality of `dsp.FarrowRateConverter` System object in Simulink.
- The Sample-Rate Converter block uses polyphase filters, which are adapted to interpolation and decimation with an integer factor, and to fractional rate conversions with a low conversion factor. This block implements the functionality of the `dsp.SampleRateConverter` System object.

Implement FIR and IIR filters in Simulink, using the Lowpass Filter and Highpass Filter blocks

- The Lowpass Filter block designs FIR or IIR low pass filters with minimum-order or specified order options. This block implements the functionality of the `dsp.LowpassFilter` System object.
- The Highpass Filter block designs FIR or IIR high pass filters with minimum-order or specified order options. This block implements the functionality of the `dsp.HighpassFilter` System object.

Estimate power spectrum and power spectral density using the Spectrum Estimator block

The Spectrum Estimator block combines the functionality of `dsp.SpectrumEstimator` System object, and the functionality included in the Spectrum Analyzer scope block, such as the ability to specify resolution bandwidth, automatic buffering with custom overlap, max-hold and min-hold spectra, and power units.

Automatic selection of filter coefficients for FIR Interpolation, FIR Decimation, and FIR Rate Conversion blocks

The FIR Interpolation, FIR Decimation, and FIR Rate Conversion blocks can now choose their filter coefficients automatically. In the block dialog box, under **Coefficient source**, select `Auto`. The block then uses the coefficients of an FIR Nyquist filter, designed for the rate conversion factor of the block. For more information, see the 'Choose Filter Coefficients Automatically' section in the block reference page.

This feature is supported for HDL code generation from the FIR Interpolation and FIR Decimation blocks. HDL code generation requires an HDL Coder license.

Visualize the frequency response of the underlying filters in the DSP System Toolbox blocks

You can now use FVTool to visualize the frequency response of the filter in these filter blocks:

- DC Blocker
- Variable Bandwidth FIR Filter and Variable Bandwidth IIR Filter
- Notch-Peak Filter
- Parametric EQ Filter
- Digital Down-Converter and Digital Up-Converter
- Farrow Rate Converter
- Sample-Rate Converter
- FIR Halfband Interpolator and FIR Halfband Decimator
- IIR Halfband Interpolator and IIR Halfband Decimator

-
- CIC Compensation Interpolator and CIC Compensation Decimator
 - Lowpass Filter and Highpass Filter

In the block dialog box, click the **View Filter Response** button. FVTool opens and displays the filter frequency response, which is computed based on the block dialog box parameters. Changes made to these parameters update the FVTool response. For more information, see the 'View Filter Response' section in the block reference page.

Specify the window length and window overlap in Cross-Spectrum Estimator and Discrete Transfer Function Estimator blocks

Cross-Spectrum Estimator block and Discrete Transfer Function Estimator block now accept **Window length** and **Window Overlap** parameters. These parameters buffer the input data into overlapping segments, enabling you to implement the Welch spectrum estimation algorithm.

Select the color of the noise in Colored Noise block

In the Colored Noise block, you can now specify the color of the noise to generate by setting the **Noise color** parameter to:

- White
- Pink
- Brown
- Blue
- Purple
- Custom

When you choose `Custom`, you can specify the power density of the noise through the **Power of inverse frequency** parameter.

New functionality added to the `dsp.SpectrumEstimator` System object

`dsp.SpectrumEstimator` System object now has these properties, which are also found in the Spectrum Analyzer:

- `PowerUnits` — Units in which the `dsp.SpectrumEstimator` displays the power values, specified as `dBm`, `dBW`, or `Watts`.

- `ReferenceLoad` — Reference load, in ohms, that `dsp.SpectrumEstimator` uses as a reference to compute the power values.
- `OutputMaxHoldSpectrum` — The maximum-hold spectrum at each frequency bin. To compute this value, `dsp.SpectrumEstimator` keeps the maximum value of all the power spectrum estimates.
- `OutputMinHoldSpectrum` — The minimum-hold spectrum at each frequency bin. To compute this value, `dsp.SpectrumEstimator` keeps the minimum value of all the power spectrum estimates.

Generate C code from `dsp.AllpassFilter` and import the System object into Simulink using the MATLAB System block

When the `Structure` property of `dsp.AllpassFilter` System object is set to `Minimum multiplier` or `Lattice`, you can:

- Generate C code from the System object.
- Import this System object into Simulink using MATLAB System block.
- Specify the last section of the filter as first order by setting the `TrailingFirstOrderSection` property to `true`.
- Assign array data to the `AllpassCoefficients` and `LatticeCoefficients` properties.

When the `Structure` property is set to `Wave Digital Filter`, the `WDFCoefficients` property accepts cell arrays only. This configuration does not support code generation and cannot interface with the MATLAB System block.

Compatibility Considerations

Old MATLAB code with `AllpassCoefficients` and `LatticeCoefficients` properties set to a cell array, continues to run in R2015b. However, it is recommended that you set these properties to an array value. The option to set the `Structure` property to `Wave Digital Filter` will be removed in a future release.

dsp.CICDecimator and dsp.CICInterpolator System objects support single and double data types

Inputs and outputs to the `dsp.CICDecimator` and `dsp.CICInterpolator` System objects can now be single or double data types, in addition to the fixed-point data type.

Frame-based signal logging in structure formats in Time Scope block

For frame-based, single-port signals and multiport signals, the Time Scope block now supports logging in the `Structure with Time` format and `Structure` formats.

Scientific notation in Time Scope

The Time Scope block and System object now uses scientific E notation in the measurement panels. Previously they displayed SI units.

Performance improvements for FFT, IFFT and notch peak filters

Simulation speed has improved for:

- `dsp.FFT`, `dsp.IFFT`, and `dsp.NotchPeakFilter` System objects.
- FFT and IFFT blocks.

Floating-point support and optional valid port for HDL-optimized NCO

The NCO HDL Optimized block and the `dsp.HDLNCO` System object now support floating-point (`double` or `single`) input data for use with Fixed-Point Designer™ tools.

When a data input is fixed point, or when no data input ports are enabled, the block computes a fixed-point output waveform based on the fixed-point parameters. When a data input is floating-point, the block ignores the fixed-point parameters, and computes a double-precision output waveform.

The `validIn` port on the NCO HDL Optimized block and the `validIn` argument to the `step` method of the `dsp.HDLNCO` System object are now optional. This port or argument is enabled by default.

HDL Code Generation from filterbuilder

Using Filter Design HDL Coder™, you can generate HDL code from the filter objects designed in `filterbuilder`. On the **Code Generation** tab, click **Generate HDL** to set HDL code generation options and generate code. You can generate HDL code from the following filter types:

Filter	Structure	System object
FIR	Direct form Direct form transposed Direct form symmetric Direct form antisymmetric	<code>dsp.FIRFilter</code>
FIR Decimator	Direct form Direct form transposed	<code>dsp.FIRDecimator</code>
FIR Interpolator	Direct form Direct form transposed	<code>dsp.FIRInterpolator</code>
IIR	Direct form I Direct form I transposed Direct form II Direct form II transposed	<code>dsp.BiquadFilter</code>
CIC Decimator		<code>dsp.CICDecimator</code>
CIC Interpolator		<code>dsp.CICInterpolator</code>

Simulink templates for ARM Cortex-A and ARM Cortex-M processors

Simulink model templates for ARM Cortex-A and ARM Cortex-M processors have been added to the template gallery. These templates enable the reuse of settings, including configuration parameters for models with optimized code generation for ARM Cortex-A or ARM Cortex-M processors. For successful code generation, the provided model requires DSP System Toolbox Support Package for ARM Cortex-A Processors or DSP System Toolbox Support Package for ARM Cortex-M Processors. Instead of using the new model default canvas, select a template model to get started. The templates create models that use best practices and previous solutions to common problems. To avoid having to reconfigure your model for your environment or application, use a provided template or create templates from your existing models.

For more information, see [Configure the Simulink Environment for Signal Processing Models](#).

ROI processing removed

The **Enable ROI processing** parameter has been removed from the following blocks in the `dspstat3` library. In addition, the `ROIProcessing` property has been removed from the corresponding System objects.

- Maximum block and `dsp.Maximum` System object
- Minimum block and `dsp.Minimum` System object
- Mean block and `dsp.Mean` System object
- Standard Deviation block and `dsp.StandardDeviation` System object
- Variance block and `dsp.Variance` System object

If you have Computer Vision System Toolbox™ software installed, use the equivalent block from the `visionstatistics` library. To select the rectangular region, use the Simulink Selector block.

Compatibility Considerations

Blocks in old models implementing ROI processing no longer use this functionality in R2015b. To disable ROI processing from these models, at the MATLAB command prompt, enter `set_param(blockpath, 'roiEnable', 'off')`.

Frame-based processing changes

As part of the changes in how DSP System Toolbox handles frame-based processing, certain block options have been removed.

The following sections provide more detailed information about the specific R2015b DSP System Toolbox software changes for frame-based processing:

- “Inherited Option Removed from the Input Processing Parameter” on page 5-9
- “Sample-Based Row Vector Processing Changes” on page 5-10
- “Blocks Emit Sample-Based Signals Only” on page 5-12

Inherited Option Removed from the Input Processing Parameter

Inherited option has been removed from the **Input processing** parameter in these blocks:

- FIR Interpolation
- FIR Decimation
- Two-Channel Synthesis Subband Filter
- Two-Channel Analysis Subband Filter

Compatibility Considerations

In old models that have the **Input processing** parameter set to `Inherited`, the behavior has not changed. However, it is recommended that you update this parameter to either `Columns as channels (frame based)` or `Elements as channels (sample based)`. For information on how to choose the **Input processing** parameter, see “Input processing parameter set to `Inherited`” on page 6-10.

Sample-Based Row Vector Processing Changes

In previous releases, some of the blocks that treated sample-based row vector inputs as columns had a **Treat sample-based row input as column** check box which explicitly enabled this behavior. Other blocks automatically processed sample-based row vector inputs as column vectors.

In R2015b, these blocks now treat sample-based row vector inputs as an n channel input, where n is the number of samples in the input signal.

- **Treat sample-based row input as column** check box has been removed from these blocks.
 - Maximum
 - Mean
 - Median
 - Minimum
 - Normalization
 - RMS
 - Standard Deviation
 - Variance
- These blocks no longer treat sample-based row vectors as single-channel column vectors:

-
- Autocorrelation
 - Autocorrelation LPC
 - Burg AR Estimator
 - Burg Method
 - Complex Cepstrum
 - Convolution
 - Correlation
 - Covariance AR Estimator
 - Covariance Method
 - DCT
 - FFT
 - IDCT
 - IFFT
 - Levinson-Durbin
 - LPC to LSF/LSP Conversion
 - LPC to/from Cepstral Coefficients
 - LPC to/from RC
 - LPC/RC to Autocorrelation
 - LSF/LSP to LPC Conversion
 - Modified Covariance AR Estimator
 - Modified Covariance Method
 - Polynomial Stability Test
 - Real Cepstrum
 - Yule-Walker AR Estimator
 - Yule-Walker Method

Compatibility Considerations

In R2015b, old models that treat sample-based row vector inputs as columns, might produce unexpected results. To ensure consistent results, place a Math Function block, with the **Function** parameter set to `transpose`, in front of the affected block. The Math

Function block transposes the sample-based row vector into a column vector, which is then input into the affected block.

Blocks Emit Sample-Based Signals Only

- 1 **Source blocks:** In previous releases, the following source blocks emitted frame-based signals (double lines) when `samples per frame` was greater than 1 and sample-based signals (single lines), when `samples per frame` was 1. In R2015b, the source blocks now emit sample-based signals (single lines), irrespective of the value of `samples per frame`.
 - Sine Wave
 - Chirp
 - From Audio Device
 - Random Source
 - Discrete Impulse
 - NCO
 - Signal From Workspace
 - Triggered Signal From Workspace
- 2 **Nonsource blocks:** In previous releases, the following blocks emitted frame-based signals even when the input was sample-based. In R2015b, these blocks emit only sample-based signals.
 - Buffer
 - CIC Interpolation
 - Dyadic Analysis Filter Bank
 - Dyadic Synthesis Filter Bank
 - DWT
 - IDWT
 - Inverse Short-Time FFT
 - Delay Line

Compatibility Considerations

Blocks emitting frames in models created in previous releases continue to emit frames in R2015b. To update these blocks to emit samples, use Simulink Upgrade Advisor. To

ensure consistent results with a previous release, use the Frame Conversion block after the affected block.

Features removed, replaced and renamed

Blocks removed and replaced

DSP Block Removed	Use This Instead	Backward Compatibility
Pulse Shaping Filter (with Filter Type set to Decimator and Pulse shape set to Raised Cosine or Squared Root Raised Cosine).	Raised Cosine Receive Filter in Communications System Toolbox™.	None Old models using the Pulse Shaping Filter block still run.
Pulse Shaping Filter (with Filter Type set to Interpolator and Pulse shape set to Raised Cosine or Squared Root Raised Cosine).	Raised Cosine Transmit Filter in Communications System Toolbox.	None Old models using the Pulse Shaping Filter block still run.
DSP Block Replaced	Use This Instead	Backward Compatibility
CIC Compensator (with Filter Type set to Decimator)	CIC Compensation Decimator	None Old models using the CIC Compensator block still run.
CIC Compensator (with Filter Type set to Interpolator)	CIC Compensation Interpolator	None Old models using the CIC Compensator block still run.
Halfband Filter (with Impulse response set to FIR, and Filter Type set to Decimator)	FIR Halfband Decimator	None Old models using the Halfband Filter block still run.

DSP Block Replaced	Use This Instead	Backward Compatibility
Halfband Filter (with Impulse response set to FIR, and Filter Type set to Interpolator)	FIR Halfband Interpolator	None Old models using the Halfband Filter block still run.
Halfband Filter (with Impulse response set to IIR, and Filter Type set to Decimator)	IIR Halfband Decimator	None Old models using the Halfband Filter block still run.
Halfband Filter (with Impulse response set to IIR, and Filter Type set to Interpolator)	IIR Halfband Interpolator	None Old models using the Halfband Filter block still run.

The Lowpass Filter and Highpass Filter blocks have been modified to match the functionality and interface of the `dsp.LowpassFilter` and `dsp.HighpassFilter` System objects.

Removal of `adaptfilt` objects

`adaptfilt` objects are removed and cause a warning message. In a future release, these objects will be removed entirely, so you should now use the corresponding System object. See “Removal of `adaptfilt` objects” on page 6-29 for a list of replacement objects.

Removal of `mfilt` objects

`mfilt` objects will be removed in a future release. Instead, use their System object counterparts, which are more powerful and support code generation.

<code>mfilt</code> Object	Use This System object Instead
<code>mfilt.cascade</code>	<code>dsp.FilterCascade</code>
<code>mfilt.cicdecim</code>	<code>dsp.CICDecimator</code>
<code>mfilt.cicinterp</code>	<code>dsp.CICInterpolator</code>
<code>mfilt.farrowsrc</code>	<code>dsp.FarrowRateConverter</code>

mfilt Object	Use This System object Instead
<code>mfilt.fftfilterinterp</code>	<code>dsp.FIRInterpolator</code> (approximates <code>mfilt.fftfilterinterp</code>)
<code>mfilt.firdecim</code>	<code>dsp.FIRDecimator</code>
<code>mfilt.firtdecim</code>	<code>dsp.FIRDecimator</code>
<code>mfilt.firinterp</code>	<code>dsp.FIRInterpolator</code>
<code>mfilt.firsrc</code>	<code>dsp.FIRRateConverter</code>
<code>mfilt.holdinterp</code>	<code>dsp.CICInterpolator</code> (with <code>NumSections = 1</code> , approximates <code>mfilt.holdinterp</code>)
<code>mfilt.iirdecim</code>	<code>dsp.CICInterpolator</code> <code>dsp.IIRHalfbandDecimator</code>
<code>mfilt.iirinterp</code>	<code>dsp.CICInterpolator</code> <code>dsp.IIRHalfbandInterpolator</code>
<code>mfilt.iirwdfdecim</code>	<code>dsp.IIRHalfbandDecimator</code> (approximates <code>mfilt.iirwdfdecim</code>)
<code>mfilt.iirwdfinterp</code>	<code>dsp.IIRHalfbandInterpolator</code> (approximates <code>mfilt.iirwdfinterp</code>)
<code>mfilt.linearinterp</code>	<code>dsp.CICInterpolator</code> (with <code>NumSections = 2</code> , approximates <code>mfilt.linearinterp</code>)

System Object Propagation Mixin Methods Renamed

System object propagation mixin methods have been renamed. You use propagation methods to control System object data specifications in Simulink. If you use an old method name, an error occurs. You can use `sobjupdate` to update the following renamed methods to the new methods.

Renamed Propagation Method	New Propagation Method
<code>inputComplexity</code>	<code>propagatedInputComplexity</code>
<code>outputComplexity</code>	<code>propagatedOutputComplexity</code>

Renamed Propagation Method	New Propagation Method
inputDataType	propagatedInputDataType
outputDataType	propagatedOutputDataType
inputSize	propagatedInputSize
outputSize	propagatedOutputSize
inputFixedSize	propagatedInputFixedSize
outputFixedSize	propagatedOutputFixedSize

R2015a

Version: 9.0

New Features

Bug Fixes

Compatibility Considerations

Audio Latency Reduction: Significantly reduce latency for audio hardware I/O in MATLAB and Simulink

Improvements to the audio I/O infrastructure significantly reduce latency.

To determine audio latency on your system, see [Measuring Audio Latency](#).

Filter Design Enhancements: Design high-order IIR parametric EQ filter, variable bandwidth FIR and IIR filters, Digital Down-Converter and Digital Up-Converter blocks

Implement FIR and IIR filters conveniently with minimal design options, using the following new filter design features:

- `iirparameq` function designs IIR biquad parametric equalizer filters.
- `dsp.LowpassFilter` System object designs FIR or IIR lowpass filters with minimum-order or with specified order options.
- `dsp.HighpassFilter` System object designs FIR or IIR highpass filters with minimum-order or with specified order options.

Implement in Simulink several new filters, a power spectrum estimator, signal operations and generate colored noise:

- The Variable Bandwidth FIR Filter and Variable Bandwidth IIR Filter blocks implement the functionality of `dsp.VariableBandwidthFIRFilter` and `dsp.VariableBandwidthIIRFilter` System objects. These blocks allow you to vary the passband while filtering. Also, they enable you to tune the filter in a computationally efficient way while preserving the filter structure.
- The Parametric EQ Filter block implements a parametric equalizer with tunable gain, bandwidth, and center frequency. These filters are widely used in audio processing applications. The Parametric EQ Filter block implements the functionality of `dsp.ParametricEQFilter` System object. This block replaces the Param EQ block, found in `dspfdesign` library.
- The Notch-Peak Filter block implements a notching or peaking IIR filter in Simulink. Notch-Peak filters are used in many signal processing applications, including tone removal and removal of power line interference. With the Notch-Peak filter block, you can use tunable parameters to control the center frequencies and 3-dB bandwidths of the notches and peaks. The Notch-Peak Filter block implements the functionality of

the `dsp.NotchPeakFilter` System object. This block replaces the Peak-Notch Filter block, found in `dspfdesign` library.

- The Cross Spectrum Estimator block uses Welch's modified periodogram method to compute the cross-power spectrum of inputs. This block implements the functionality of the `dsp.CrossSpectrumEstimator` System object.
- The Digital Down-Converter and Digital Up-Converter blocks provide tools to design decimation and interpolation filters, and simplify the steps required to implement down conversion and up conversion, in Simulink. These blocks implement the functionality of the `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System objects.
- The Colored Noise block generates a colored noise signal in Simulink. With this block, you can generate noise with a $1/f^\alpha$ power spectral density. Set α equal to any value in the interval $[-2,2]$. Specifying $\alpha = 1$ results in pink noise. Specifying $\alpha = 2$ produces Brownian noise. This block implements the functionality of the `dsp.ColoredNoise` System object.

DSP Simulink Model Templates: Configure the Simulink environment for digital signal processing models

DSP Simulink model templates enable reuse of settings, including configuration parameters. Create models from templates to encourage best practices and take advantage of previous solutions to common problems. Instead of the default canvas of a new model, select a template model to help you get started. Choose `blank`, `basic` or `audio` model templates to create a skeletal model using settings recommended for DSP System Toolbox.

You can use built-in templates or create templates from models that you already configured for your environment or application.

For more information, see [Configure the Simulink Environment for Signal Processing Models](#).

Streaming Scope Improvements: Plot in stem mode, access log x-axis scaling, customize sample rate, and use infinite data support

The following improvements have been made to streaming scopes:

- Legend settable programmatically in Spectrum Analyzer block and `dsp.SpectrumAnalyzer` System object.
- Stem plot mode for Spectrum Analyzer block and System object.
- Log frequency axis for Spectrogram mode in Spectrum Analyzer block and System object.
- Vector-tunable frequency offset in Spectrum Analyzer block and System object.
- Custom sample rate in Spectrum Analyzer block.
- Variable-size support in Spectrum Analyzer System object.
- Infinite data support in Time Scope block.
- Log *x*-axis scaling in the `dsp.ArrayPlot` System object.

Library for HDL Supported DSP Blocks: Find all blocks that support HDL

In the Simulink Library Browser, use the **DSP System Toolbox HDL Support** library to find all the DSP System Toolbox blocks that support HDL code generation.

Alternatively, at the MATLAB command prompt, enter `dsphdllib` to open this library.

All blocks in this library have their parameters configured for HDL code generation. To generate HDL code, you must have an HDL Coder license.

C Code Generation of DSP Algorithms for ARM Cortex-A and Cortex-M processors: Generate optimized and faster performing C code using Embedded Coder

In R2015a, using Embedded Coder, you can generate C code optimized for ARM processors using these DSP blocks and System objects:

ARM Cortex-A processors

- `dsp.LowpassFilter` System object, with `FilterType` set to `FIR`
- `dsp.HighpassFilter` System object, with `FilterType` set to `FIR`
- Variable Bandwidth FIR Filter block

ARM Cortex-M processors

- `dsp.LowpassFilter` System object, with `FilterType` set to FIR or IIR
- `dsp.HighpassFilter` System object, with `FilterType` set to FIR or IIR
- `dsp.VariableBandwidthIIRFilter` System object, with `FilterType` set to Lowpass or Highpass
- Variable Bandwidth IIR Filter block, with `FilterType` property set to Lowpass and Highpass
- Variable Bandwidth FIR Filter block

Performance Improvements

Simulation speed has improved for:

- `dsp.FIRDecimator`, `dsp.AllpassFilter`, and `dsp.CoupledAllpassFilter` System objects
- FIR Decimation and Downsample blocks

Updated Time Scope block toolbar and menus

The following Time Scope block menu and toolbar items have been added or updated:

- **Zoom Out** option added to **Tools** menu and toolbar
- Axes scaling options grouped into an **Axes Scaling** submenu of the **Tools** menu
- **Scale Axes Limits** option indicates axes to be scaled
- **Save** and **Restore Axes Limits** added to **Axes Scaling** submenu
- Warnings appear in drop-down area above the plot, instead of in a separate dialog box

Specify block filter characteristics through System objects

You can now specify the filter characteristics of FIR Decimation, FIR Interpolation, FIR Rate Conversion, CIC Decimation, CIC Interpolation, and Biquad Filter blocks using System objects.

In the block dialog box, under **Coefficient source**, select `Filter object`. You can now specify the name of the System object in the filter object variable. See the **Specify Multirate Filter Object** section in each of these block reference pages.

Block	System object to specify
FIR Decimation	<code>dsp.FIRDecimator</code>
FIR Interpolation	<code>dsp.FIRInterpolator</code>
FIR Rate Conversion	<code>dsp.FIRRateConverter</code>
CIC Decimation	<code>dsp.CICDecimator</code>
CIC Interpolation	<code>dsp.CICInterpolator</code>

See the **Specify Discrete-Time Filter Object** section in this block reference page.

Block	System object to specify
Biquad Filter	<code>dsp.BiquadFilter</code>

This feature is supported for HDL code generation from the FIR Decimation, FIR Interpolation, CIC Decimation, CIC Interpolation, and Biquad Filter blocks. HDL code generation requires a HDL Coder license.

Discrete Transfer Function Estimator block

You can now configure the Discrete Transfer Function Estimator block to estimate the output coherence, or spectral coherence, of the input and output signal of the estimated transfer function.

In the dialog box, select the Output magnitude squared coherence estimate parameter to compute the spectral coherence between the input and output signals. Spectral coherence is a useful metric for estimating transfer functions, especially for the purposes of system identification.

Specify filter coefficients as an input to the FIR Decimation block

You can now provide filter coefficients to the FIR Decimation block. In the block dialog box, under **Coefficient Source**, select Input Port. Coefficient values are tunable (can change during simulation), while their properties must remain constant.

See the **Provide Filter Coefficients through Input port** section in the FIR Decimation block reference page.

Enhanced code generation for CIC Decimation and CIC Interpolation filter blocks

Code generated from CIC Decimation and CIC Interpolation filter blocks can now support data types that have word lengths greater than 32 bits.

HDL support for 'inherit via internal rule' data type setting on FIR Decimation and Interpolation blocks

FIR Decimation and FIR Interpolation blocks now support HDL code generation with data types specified by **Inherit via internal rule**. HDL code generation requires a HDL Coder license.

Improvements for creating System objects

The following improvements have been made to creating your own System objects:

- Number of allowable code generation inputs increased to 32
- Code generation support for unbounded variable-size vectors
- `isInputSizeLockedImpl` method for specifying whether the input port dimensions are locked
- `matlab.system.display.Action` class, used in the `getPropertyGroupsImpl` method, to define a MATLAB System block button that can call a System object method
- `getSimulateUsingImpl` and `showSimulateUsingImpl` methods to set the value of the `SimulateUsing` parameter and specify whether to show the `SimulateUsing` parameter in the MATLAB System block

Min/Max logging instrumentation for float-to-fixed-point conversion of DSP System objects

Convert the following DSP System Toolbox System objects to fixed-point using the Fixed-Point Converter app (requires a Fixed-Point Designer license):

- `dsp.FIRDecimator`
- `dsp.FIRInterpolator`

- `dsp.FIRFilter` (direct form transposed)
- `dsp.LUFactor`
- `dsp.VariableFractionalDelay`
- `dsp.Window`

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.

Provide variable-size input to the Delay System object

The `dsp.Delay` System object now supports variable-size input signals. When the input is a variable-size signal, the number of input rows can change during run time without having to call `release` method between two calls to the `step` method, while the number of channels must remain fixed.

See [What Is Variable-Size Data?](#) for information on variable-size signals .

Estimate output coherence of Transfer Function Estimator System object

You can now use `dsp.TransferFunctionEstimator` System object to estimate the magnitude squared coherence, or spectral coherence, of the input and output signals of the estimated transfer function. To compute the spectral coherence, specify the `OutputCoherence` property of the System object as `true`. Spectral coherence is a useful metric for estimating transfer functions, especially for the purposes of system identification.

Specify filter coefficients as an input to the FIR Decimator System object

Provide filter coefficients to the `dsp.FIRDecimator` System object as an input argument. Specify `NumeratorSource` property of the System object as one of `Property` (default) or `Input port`. When you specify `Input port`, the filter object requires the numerator coefficients to be specified as the third argument at every step.

See the `NumeratorSource` property of `dsp.FIRDecimator` for details.

Bit growth to avoid overflow in HDL-optimized FFT and IFFT

When $1/N$ scaling is disabled, the FFT algorithm grows the internal word length by 1 bit after each butterfly stage. This adjustment avoids overflow. This change affects the following blocks and System objects:

- FFT HDL Optimized
- IFFT HDL Optimized
- `dsp.HDLFFT`
- `dsp.HDLIFFT`

Fixed-point support for FIR Half-band Interpolator and FIR Half-band Decimator System objects

Implement HDL focused fixed-point operations on `dsp.FIRHalfbandInterpolator` and `dsp.FIRHalfbandDecimator` System objects.

Updated cost method for filter System objects

The `cost` method for filter System objects is now a structure with the following fields:

- `NumCoefficients`
- `NumStates`
- `MultiplicationsPerInputSample`
- `AdditionsPerInputSample`

Frame-based processing

As part of general product-wide changes pertaining to Frame-Based processing, certain block options that use the frame attribute of the input signal now cause an error.

The following sections provide more detailed information about the specific R2015a DSP System Toolbox software changes for frame-based processing:

- “Input processing parameter set to Inherited” on page 6-10
- “Rate options parameter set to Inherit from input” on page 6-24
- “Treat Mx1 and unoriented sample-based signals as parameter set to M channels” on page 6-25
- “Save 2-D signals as parameter set to Inherit from input” on page 6-25
- “Find the histogram over parameter set to Inherited” on page 6-26
- “Sample-based processing parameter set to Pass through” on page 6-26
- “Running difference parameter set to Inherit from input” on page 6-27

Input processing parameter set to Inherited

Setting **Input processing** parameter to `Inherited` now causes an error in these blocks:

- Biquad Filter
- FIR Interpolation
- FIR Decimation
- Edge Detector
- Analytic Signal
- Variable Integer Delay
- Variable Fractional Delay
- Zero Crossing
- Two-Channel Analysis Subband Filter
- Two-Channel Synthesis Subband Filter
- CIC Interpolation
- Upsample
- Downsample
- Repeat
- Unwrap
- Minimum, when you set **Mode** to Running
- Maximum, when you set **Mode** to Running
- Mean, when you select **Running mean** check box
- Standard Deviation, when you select **Running standard deviation** check box

-
- Variance, when you select **Running variance** check box
 - RMS, when you select **Running RMS** check box
 - Cumulative Sum, when you set **Sum input along** to Channels (running sum)
 - Cumulative Product, when you set **Multiply input along** to Channels (running product)

Compatibility Considerations

To ensure consistent results for models created in previous releases, set **Input processing** to:

- Columns as channels (frame based), for frame-based input signals (double-line)
- Elements as channels (sample based), for sample-based input signals (single-line)

After compiling the model, frame-based signals appear as double lines. Sample-based signals appear as single lines.

For models created in R2015a:

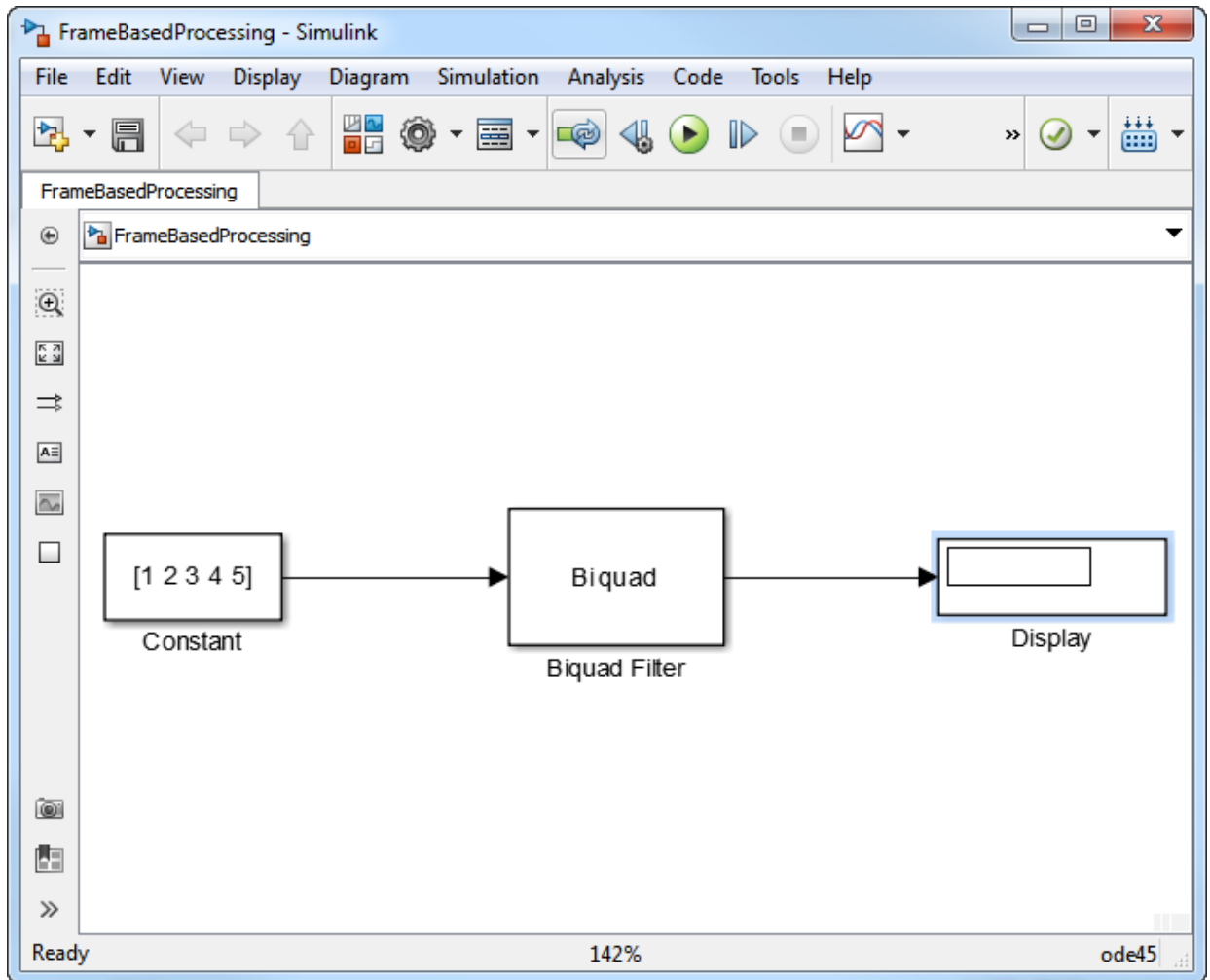
- To treat each column of the input signal as an independent channel, set **Input processing** to Columns as channels (frame based).
- To treat each element of the input signal as an independent channel, set **Input processing** to Elements as channels (sample based).

Simulink Upgrade Advisor

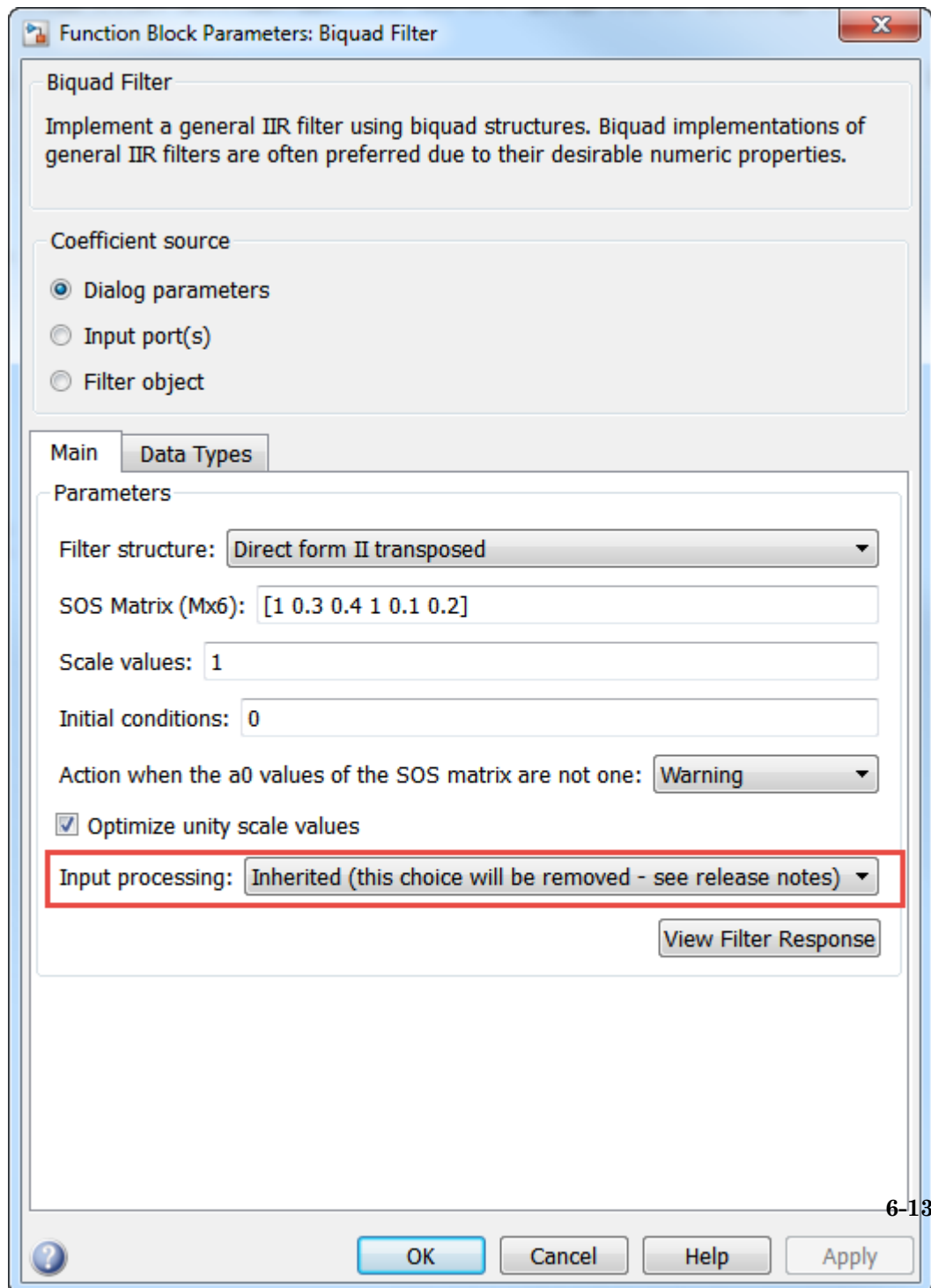
If you are not sure which **Input processing** option applies to your model and choose Inherited instead, use the Simulink Upgrade Advisor to update your model.

1 Update blocks in a model

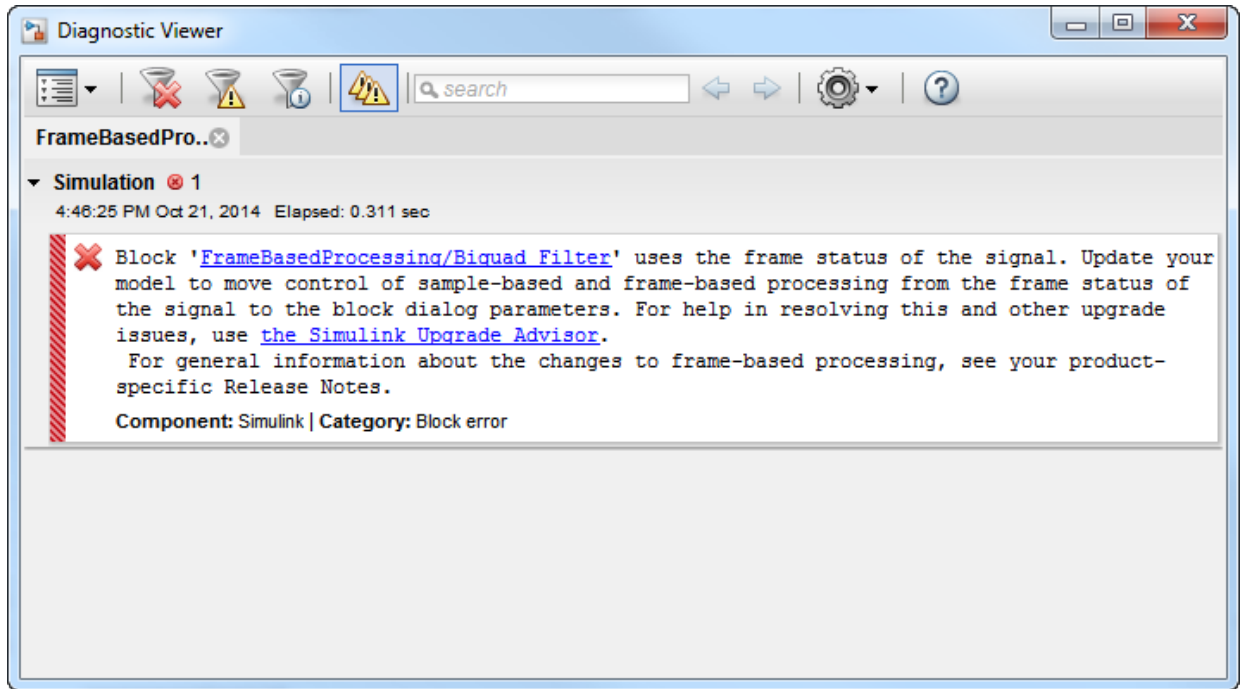
In this model, the Biquad Filter block uses frame-based processing.



In the Biquad Filter dialog box, **Input processing** is set to Inherited.

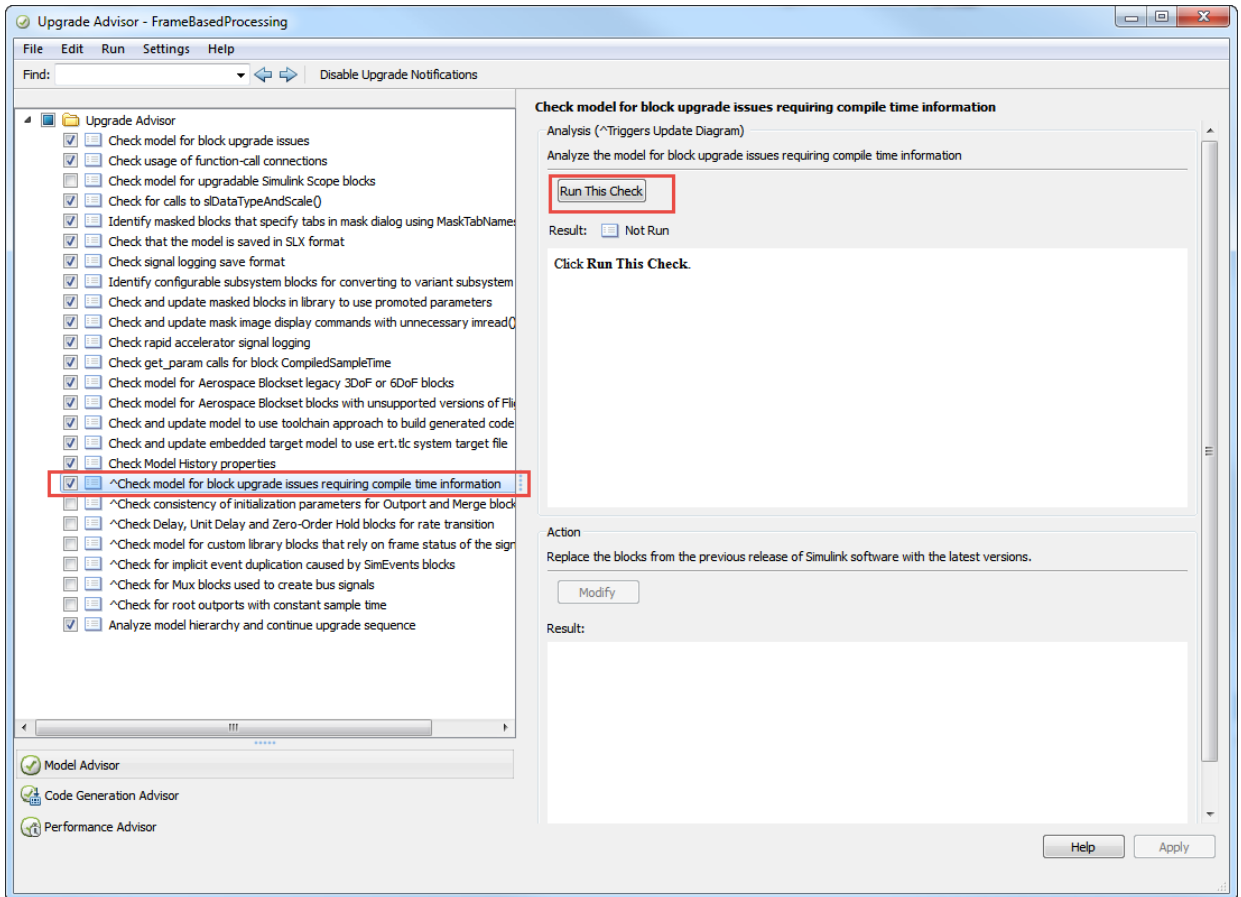


When you run the model, Simulink issues an error message.

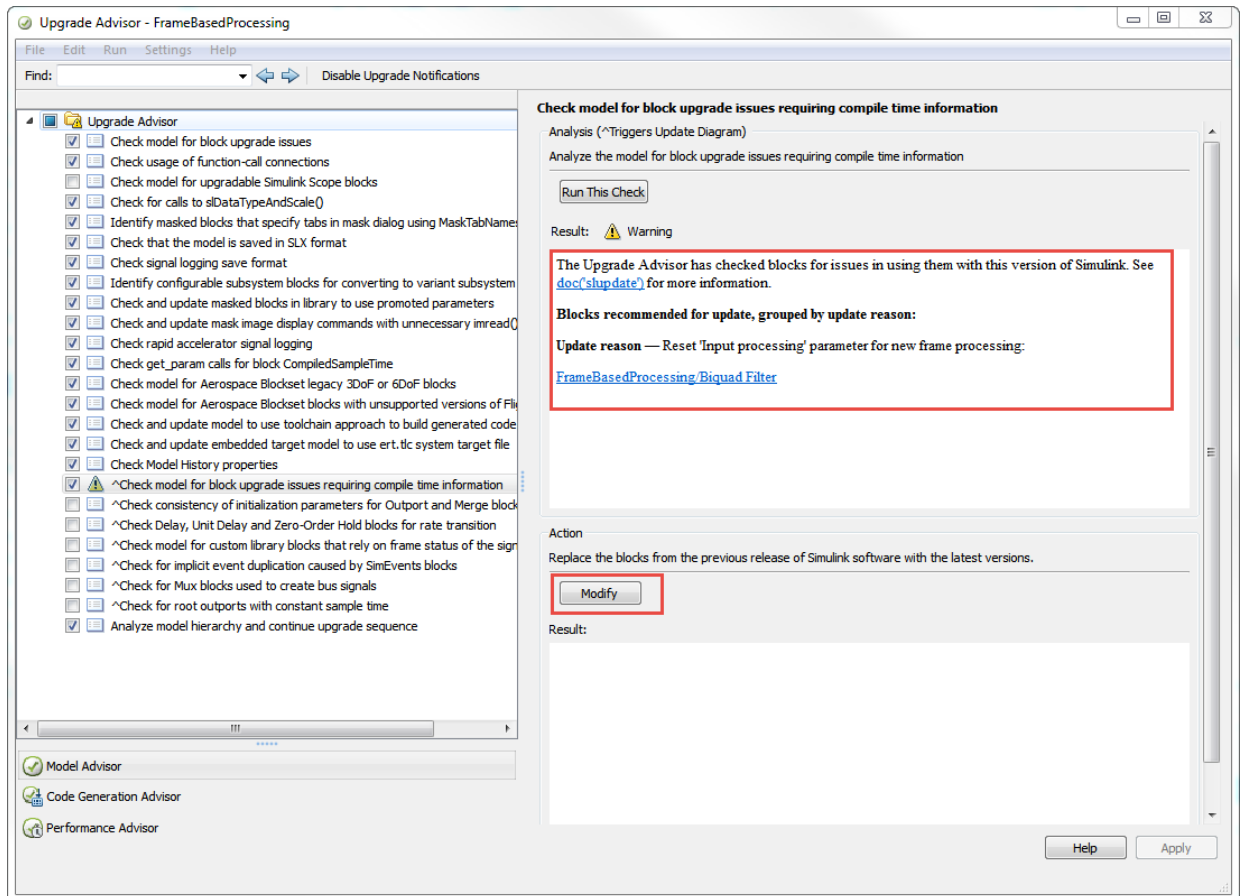


From the error message, you can open the Simulink Upgrade Advisor by clicking the hyperlink.

Then you can select the Check model for block upgrade issues requiring compile time information check and click **Run this Check**.



The Upgrade Advisor runs the check on all the blocks in the model, recommends an update for the needed blocks, and gives the reason for the update.



Click **Modify** to update the `FrameBasedProcessing/Biquad Filter` block.

Because the input signal is sample-based, in the Biquad Filter block, the Upgrade Advisor changes **Input processing** to `Elements as Channels (Sample based)`.

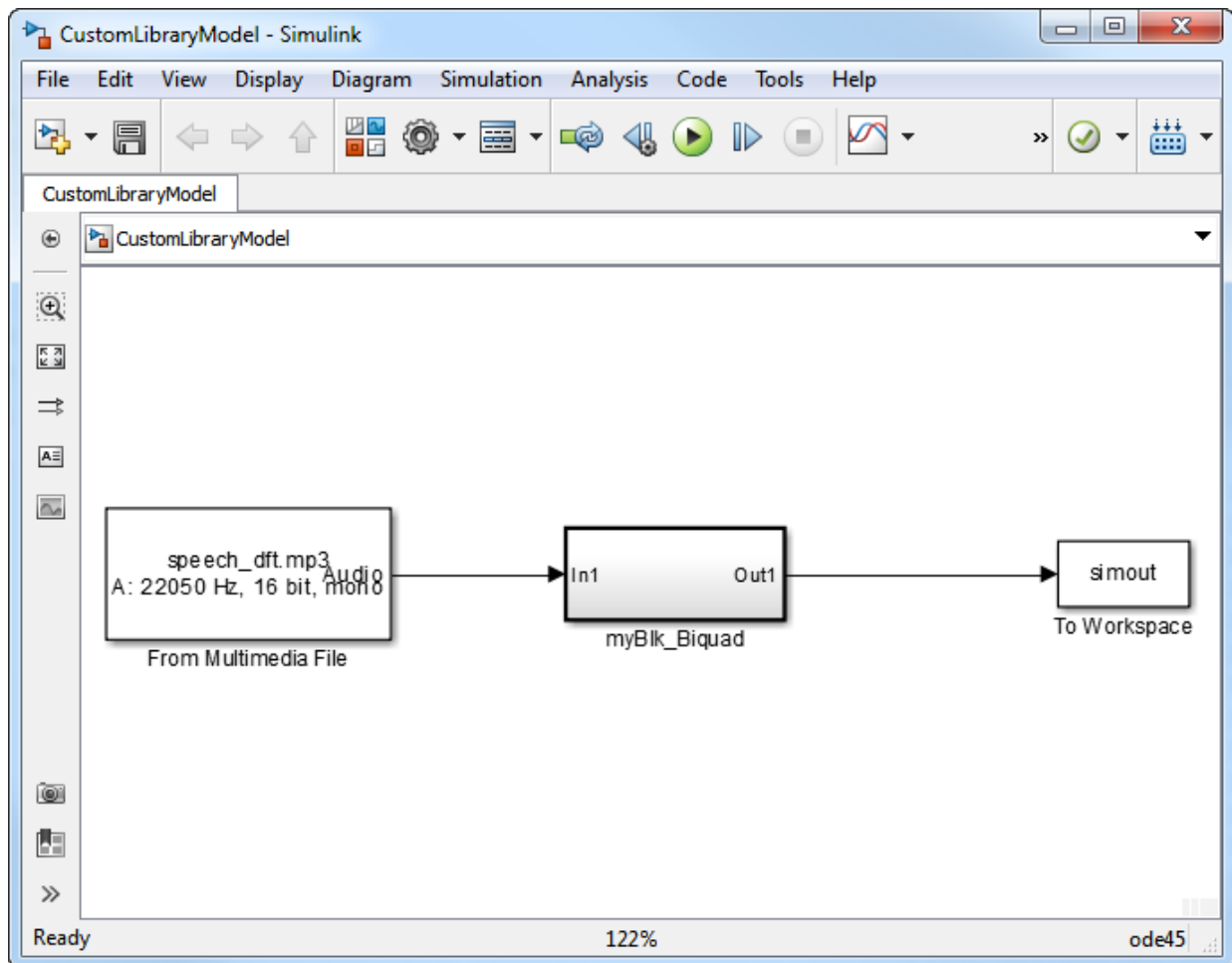
2 Update blocks in a custom library

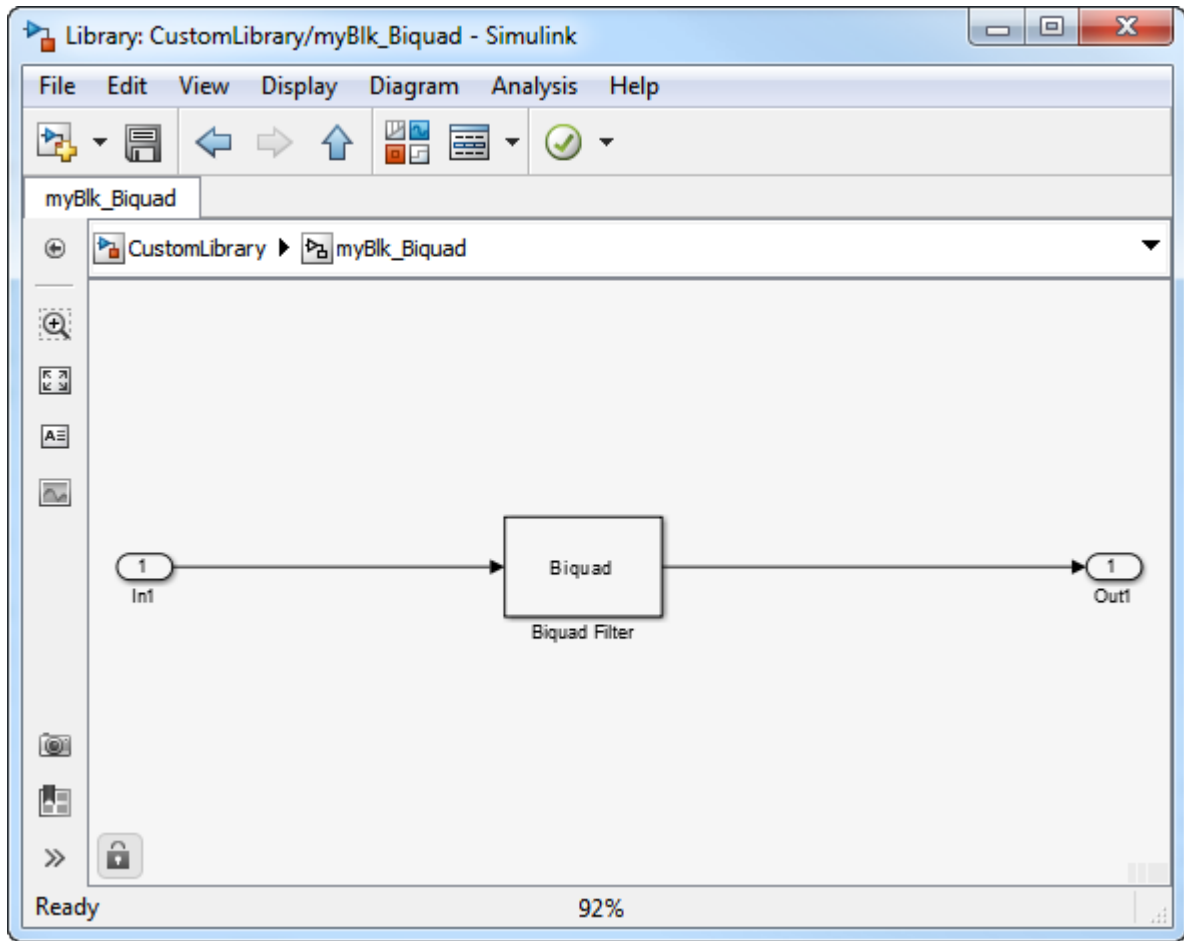
When you select the Check model for block upgrade issues requiring compile time information check box, the Upgrade Advisor does not update blocks in custom libraries.

Custom libraries are Simulink block libraries that you can create to reuse your blocks and subsystems in one or more models. For more information, see [Create Custom Block Libraries](#).

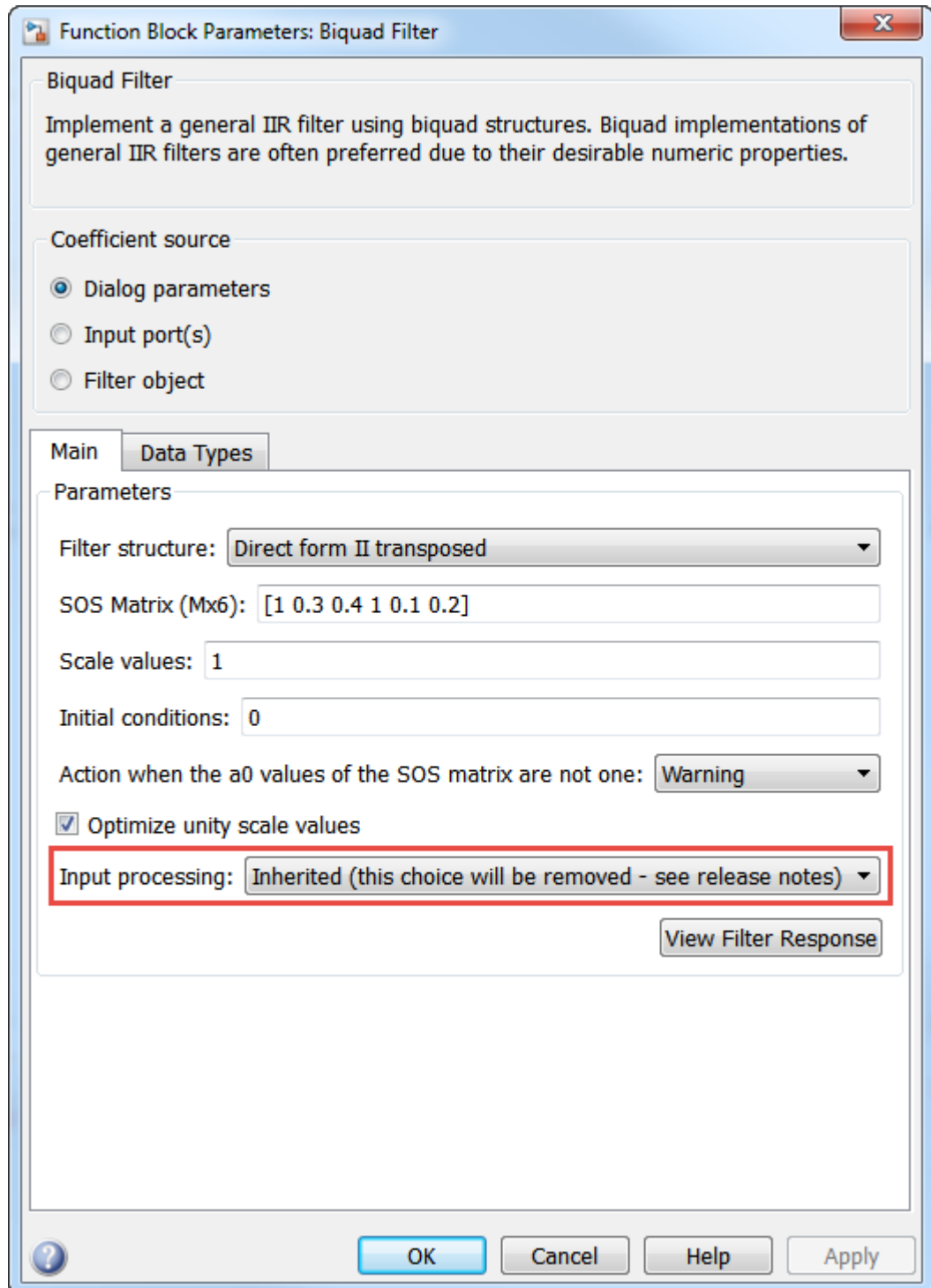
To analyze frame-based processing related errors in custom library blocks, open the Simulink Upgrade Advisor and select the `Check model for custom library blocks that rely on frame status of the signal`, and click **Run This Check**. This check does not update the library blocks. It analyzes the blocks, recommends fixes, and gives reasons for the fixes. You must make the fixes manually.

In this model, `myBlk_Biquad` is a block from the custom library. It contains a Biquad Filter block, which is one of the blocks that causes an error when you set **Input processing** to `Inherited`.

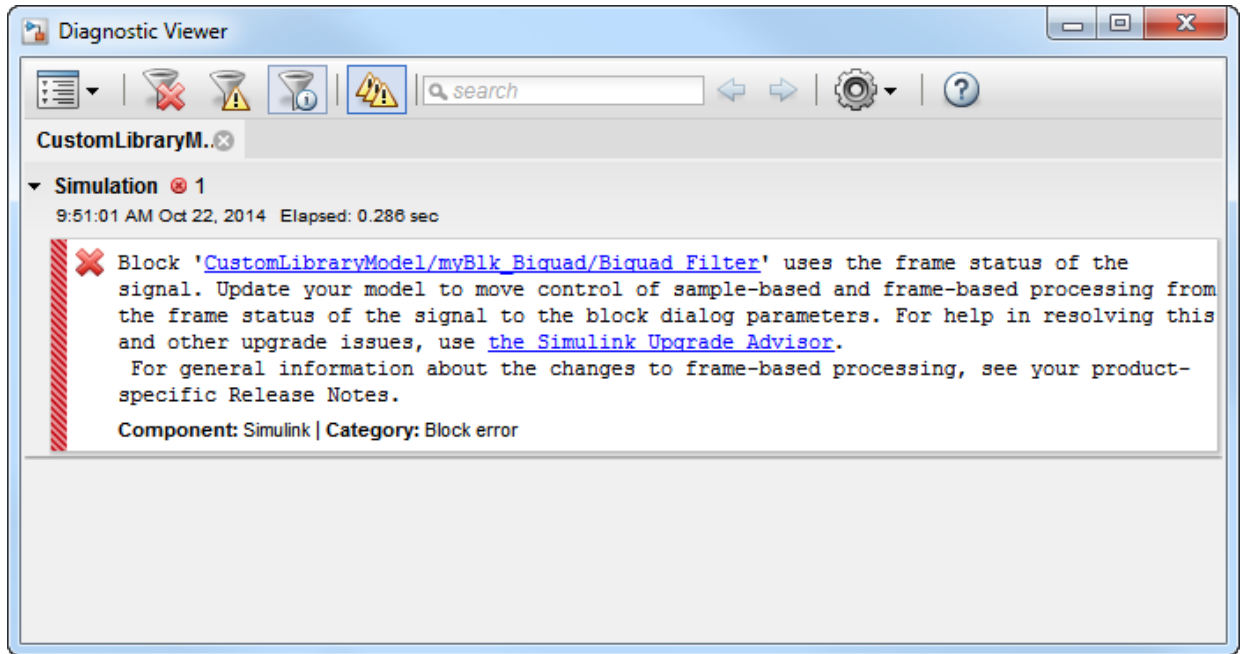




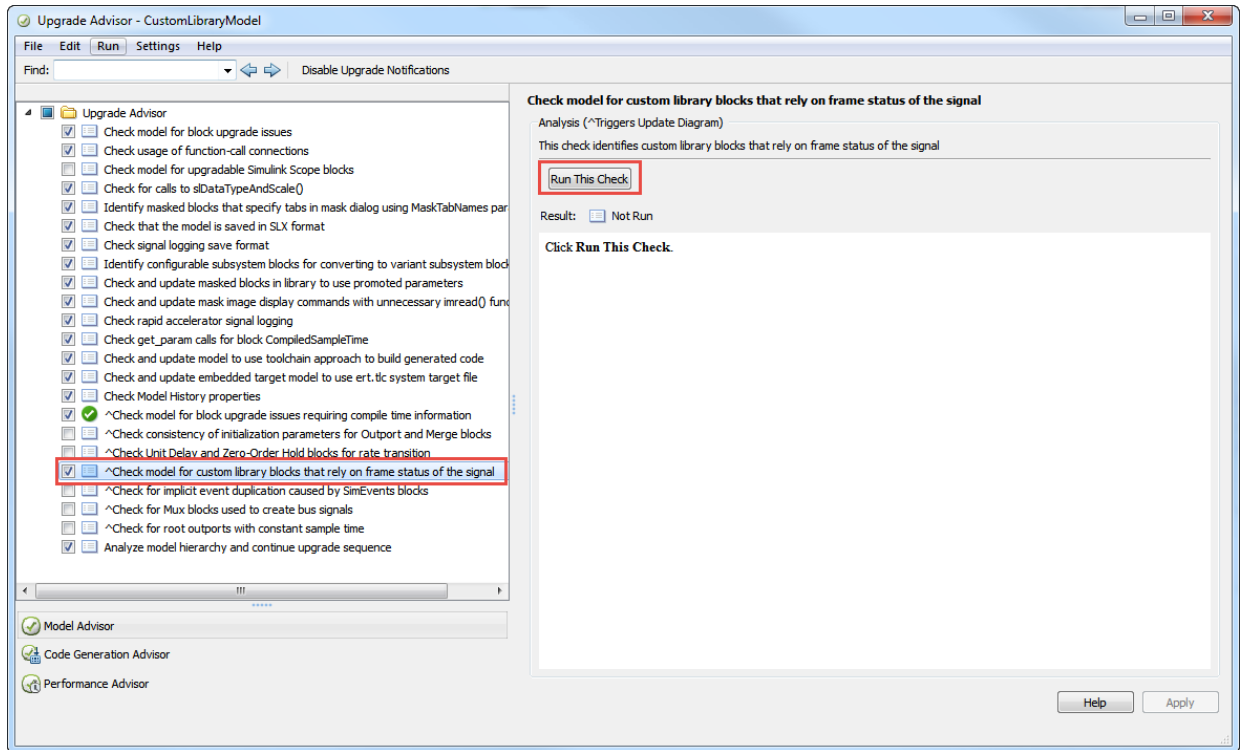
In the Biquad Filter dialog box, **Input processing** is set to Inherited.



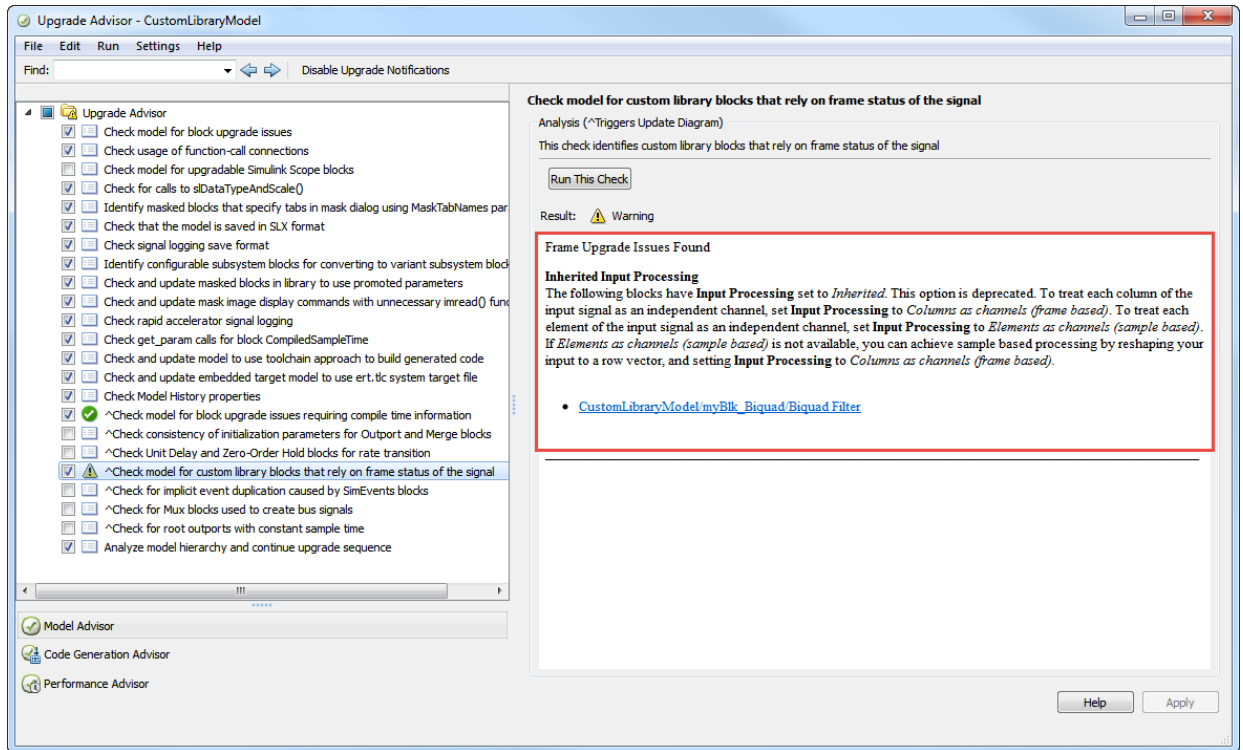
When you run the model, Simulink issues an error message.



To resolve this error message, reopen Simulink Upgrade Advisor, select the Check model for custom library blocks that rely on frame status of the signal check and click **Run This Check**.



The Upgrade Advisor runs the check on all the blocks in the custom library subsystem block, recommends an update for the needed blocks, and gives the reason for the update.



For the CustomLibraryModel/myBlk_Biquad/Biquad Filter block, the Upgrade Advisor recommends changing **Input Processing** to either Elements as channels (sample based) or Columns as channels (frame based). To make the change, you update the parameter in the library block, myBlk_Biquad/Biquad Filter, not in the specific instance of the block in the model. The model now runs successfully.

If the custom library block (in this case, myBlk_Biquad block) is used for frame-based processing in all the models that use it, set the **Input Processing** parameter to Columns as channels (frame based) in the library block. Else, set this parameter to Elements as channels (sample based).

If the library block is used for frame-based processing in some models and sample-based processing in others, you can add a parameter to the mask of the library block and configure this parameter to choose one of Columns as channels (frame

based) or `Elements as channels (sample based)`. Choosing the appropriate option in the block mask should set the **Input Processing** parameter of underlying block to `Columns as channels (frame based)` or `Elements as channels (sample based)`.

Another approach is to promote the option chosen for **Input Processing** parameter from underlying block (in this case, Biquad Filter block) to the mask of the library block. This is known as parameter promotion. For details on parameter promotion, see [Promote Underlying Block Parameters to Mask](#).

Rate options parameter set to Inherit from input

In the CIC Decimation block, setting the **Rate options** parameter to `Inherit from input` causes an error.

Compatibility Considerations

To ensure consistent results for models created in previous releases, set **Rate options** to:

- `Allow multirate processing, for sample-based input signals`
- `Enforce single-rate processing, for frame-based input signals`

For models created in R2015a:

- To run the block in single-rate mode, set **Rate options** to `Enforce single-rate processing`.
- To run the block in multirate mode, set **Rate options** to `Allow multirate processing`.

If you are not sure which option to choose, run these Simulink Upgrade Advisor checks:

- Check model for block upgrade issues requiring compile time information, for blocks in a model
- Check model for custom library blocks that rely on frame status of the signal, for blocks in a custom library

Treat Mx1 and unoriented sample-based signals as parameter set to M channels

Setting **Treat Mx1 and unoriented sample-based signals as** parameter to `M` channels now causes an error in these blocks:

- Buffer
- Delay Line
- Overlap-Save FFT Filter
- Overlap-Add FFT Filter
- Short-Time FFT

Compatibility Considerations

To ensure consistent results for models created in previous releases, reshape the input to be a 1-by- M vector, and set **Treat Mx1 and unoriented sample-based signals as** to `One channel`.

Save 2-D signals as parameter set to Inherit from input

In the `Triggered To Workspace` block, setting the **Save 2-D signals as** parameter to `Inherit from input` now causes an error.

Compatibility Considerations

To ensure consistent results for models created in previous releases, set **Save 2-D signals as** to

- **3-D array (concatenate along third dimension)**, for sample-based input signals
- **2-D array (concatenate along first dimension)**, for frame-based input signals

For models created in R2015a:

- For frame-based processing, set **Save 2-D signals as** to **2-D array (concatenate along first dimension)**.
- For sample-based processing, set **Save 2-D signals as** to **3-D array (concatenate along third dimension)**.

If you are not sure which option to choose, run these Simulink Upgrade Advisor checks:

- Check model for block upgrade issues requiring compile time information, for blocks in a model
- Check model for custom library blocks that rely on frame status of the signal, for blocks in a custom library

Find the histogram over parameter set to Inherited

In the Histogram block, setting the **Find the histogram over** to Inherited now causes an error.

Compatibility Considerations

To ensure consistent results for models created in previous releases, set **Find the histogram over** to:

- **Entire input**, for sample-based input signals
- **Each column**, for frame-based input signals

For models created in R2015a:

- To compute the histogram for each column of the input independently, set **Find the histogram over** to **Each column**.
- To compute the histogram over the entire input, set the **Find the histogram over** to **Entire input**.

If you are not sure which option to choose, run these Simulink Upgrade Advisor checks:

- Check model for block upgrade issues requiring compile time information, for blocks in a model
- Check model for custom library blocks that rely on frame status of the signal, for blocks in a custom library

Sample-based processing parameter set to Pass through

In the Unbuffer block, setting the **Sample-based processing** to Pass through now causes an error.

Compatibility Considerations

Unbuffer the M -by- N matrix input into a 1-by- N output vector by setting **Sample-based processing** to `Same as frame based`.

If you want the input to pass through, remove the Unbuffer block from the model.

Running difference parameter set to Inherit from input

In the Difference block, setting the **Running difference** to `Inherit from input` now causes an error.

Compatibility Considerations

To ensure consistent results for models created in older releases, set **Running difference** to:

- `No`, for sample-based input signals
- `Yes`, for frame-based input signals

For models created in R2015a:

- To compute the difference between adjacent elements in the current input, set **Running difference** to `No`.
- To compute the running difference across consecutive inputs, set **Running difference** to `Yes`.

If you are not sure which option to choose, run these Simulink Upgrade Advisor checks:

- Check model for block upgrade issues requiring compile time information, for blocks in a model
- Check model for custom library blocks that rely on frame status of the signal, for blocks in a custom library

Features removed, replaced, and duplicated

Blocks replaced, removed, and available in additional libraries

DSP Block Replaced	Replaced With	Backward Compatibility — What Happens When You Run Models Containing This Block?
Delay	Delay block in Simulink library	When there is an exact match in functionality, the Simulink Delay block replaces the DSP Delay block automatically.
Param EQ	Parametric EQ Filter	None Old models using the Param EQ block still run.
Peak-Notch Filter	Notch-Peak Filter	None Old models using Peak-Notch Filter block still run.
DSP Block Removed	Use This Block Instead	Backward Compatibility
Transpose	Math Function block in Simulink library	None The Math Function block replaces the Transpose block automatically.
Complex Exponential	Trigonometric Function block in Simulink library	None The Trigonometric Function block replaces the Complex Exponential block automatically.
Constant Diagonal Matrix	Constant block in Simulink library	None The Constant block replaces the Constant Diagonal Matrix block automatically.

DSP Block Available in New Library	
NCO and NCO HDL Optimized block	Blocks are now available in dspsrcs4 library in addition to dspsigops library.

Removal of adaptfilt objects

adaptfilt objects will be removed in a future release. Instead, use their System object counterparts, which are more powerful and support code generation.

adaptfilt Object	Use This System object Instead
adaptfilt.lms	dsp.LMSFilter
adaptfilt.nlms	
adaptfilt.se	
adaptfilt.sd	
adaptfilt.ss	
adaptfilt.blms	dsp.BlockLMSFilter
adaptfilt.rls	dsp.RLSFilter
adaptfilt.qdrls	
adaptfilt.swrls	
adaptfilt.hrls	
adaptfilt.hswrls	
adaptfilt.ftf	dsp.FastTransversalFilter
adaptfilt.swftf	
adaptfilt.ap	dsp.AffineProjectionFilter
adaptfilt.apru	
adaptfilt.bap	

adaptfilt Object	Use This System object Instead
adaptfilt.gal adaptfilt.lsl adaptfilt.qrdlsl	dsp.AdaptiveLatticeFilter
adaptfilt.filtxlms	dsp.FilteredXLMSFilter
adaptfilt.fdaf adaptfilt.ufdaf	dsp.FrequencyDomainAdaptiveFilter
adaptfilt.blmsfft	Will be removed in a future release
adaptfilt.adjlm adaptfilt.dlms	Will be removed in a future release
adaptfilt.pbfdaf adaptfilt.pbufdaf	Will be removed in a future release
adaptfilt.tdafdct adaptfilt.tfafdft	Will be removed in a future release

Functionality changed or being removed for blocks and System objects

Removal of sample mode from the DSP System Toolbox System objects

The `FrameBasedProcessing` property of all DSP System objects will be removed in a future release. System objects containing this property will then work only in frame-based processing mode. See [What Is Frame-Based Processing?](#) for more information. Effective R2015a, modifying this property throws a warning for these System objects:

- `dsp.AllpoleFilter`
- `dsp.AnalyticSignal`
- `dsp.BiquadFilter`
- `dsp.Buffer`
- `dsp.CumulativeProduct`
- `dsp.CumulativeSum`

-
- `dsp.Delay`
 - `dsp.FIRFilter`
 - `dsp.IIRFilter`
 - `dsp.MatFileReader`
 - `dsp.MatFileWriter`
 - `dsp.Maximum`
 - `dsp.Mean`
 - `dsp.Minimum`
 - `dsp.PeakToPeak`
 - `dsp.PeakToRMS`
 - `dsp.PhaseUnwrapper`
 - `dsp.RMS`
 - `dsp.SignalSink`
 - `dsp.StandardDeviation`
 - `dsp.VariableFractionalDelay`
 - `dsp.VariableIntegerDelay`
 - `dsp.Variance`
 - `dsp.ZeroCrossingDetector`

In the `dsp.CumulativeProduct` and `dsp.CumulativeSum` System objects, the default value of `FrameBasedProcessing` property is `true`.

Option to specify filter coefficients from Digital Up Converter and Digital Down Converter System objects being removed

`FilterSpecification` and its related properties will be removed in a future release from the `dsp.DigitalUpConverter` and `dsp.DigitalDownConverter` System objects. The System objects then will not allow you to specify coefficients for individual stages.

System object	Properties Being Removed
dsp.DigitalUpConverter	<ul style="list-style-type: none"> • FilterSpecification • FirstFilterCoefficients • SecondFilterCoefficients • FirstFilterCoefficientsDataType • SecondFilterCoefficientsDataType • CustomFirstFilterCoefficientsDataType • CustomSecondFilterCoefficientsDataType
dsp.DigitalDownConverter	<ul style="list-style-type: none"> • FilterSpecification • SecondFilterCoefficients • ThirdFilterCoefficients • SecondFilterCoefficientsDataType • ThirdFilterCoefficientsDataType • CustomSecondFilterCoefficientsDataType • CustomThirdFilterCoefficientsDataType

Removal of OutputDataType and OverflowAction properties for CIC Compensation Interpolator and Decimator System objects

The OutputDataType and OverflowAction properties for dsp.CICCompensationInterpolator and dsp.CICCompensationDecimator System objects have been removed. The OutputDataType property of these System objects is now always Same word length as input. For these System objects, the word length matches the input word length, and the fraction length is computed to give the best possible precision to the data. You no longer have access to this parameter.

R2014b

Version: 8.7

New Features

Compatibility Considerations

Optimized C code generation for ARM Cortex-A Ne10 library from MATLAB and Simulink with DSP System Toolbox Support Package for ARM Cortex-A Processors

This release adds code-generation support for ARM Cortex-A processors in MATLAB for select blocks and System objects. With the supported blocks and System objects, you can generate optimized C code that calls the Ne10 library function and compiles to provide an executable to run on ARM Cortex-A processors. To use the DSP System Toolbox Support Package for ARM Cortex-A Processors, you must have the following products:

- DSP System Toolbox
- Embedded Coder
- MATLAB Coder

To design in Simulink, you must also have these products:

- Simulink
- Simulink Coder

The following DSP System Toolbox blocks and System objects support the Ne10 library:

- Discrete FIR Filter
- FIR Decimation
- FIR Interpolation
- FFT
- IFFT
- `dsp.FIRFilter`
- `dsp.FIRDecimator`
- `dsp.FIRInterpolator`
- `dsp.FFT`
- `dsp.iFFT`
- `dsp.VariableBandwidthFIRFilter`
- `dsp.FIRHalfbandInterpolator`
- `dsp.FIRHalfbandDecimator`
- `dsp.CICCompensationDecimator`

-
- `dsp.CICCompensationInterpolator`
 - `dsp.DigitalDownConverter`
 - `dsp.DigitalUpConverter`
 - `dsp.SampleRateConverter`

For more information, see [Support Package for ARM Cortex-A Processors](#).

System objects for DSP System Toolbox Support Package for ARM Cortex-M Processors

This release adds support to generate optimized C code for the following System objects on ARM Cortex-M processors

- `dsp.VariableBandwidthFIRFilter`
- `dsp.FIRHalfbandInterpolator`
- `dsp.FIRHalfbandDecimator`
- `dsp.CICCompensationDecimator`
- `dsp.CICCompensationInterpolator`
- `dsp.DigitalDownConverter`
- `dsp.DigitalUpConverter`
- `dsp.SampleRateConverter`

Fixed-point support for Biquad Filter on DSP System Toolbox Support Package for ARM Cortex-M Processors

This release adds fixed-point support to generate optimized C code for the Biquad Filter block and `dsp.BiquadFilter` System object on ARM Cortex-M processors. The supported data formats are Q15 and Q31.

Multirate filters: Sample and Farrow Rate Converter, CIC Compensation Interpolator/Decimator, and FIR Halfband Interpolator/Decimator System objects

This release adds the following multirate filter System objects:

- `dsp.SampleRateConverter`
- `dsp.FarrowRateConverter`
- `dsp.CICCompensationInterpolator`
- `dsp.CICCompensationDecimator`
- `dsp.FIRHalfbandInterpolator`
- `dsp.FIRHalfbandDecimator`

Tunable coefficients and variable-size input available on FIR Interpolator System object and block

The FIR Interpolation block and the `dsp.FIRInterpolator` System object now support tunable coefficients and variable-size input, enabling you to specify filter coefficients from the input port.

Variable-size input available on FIR Decimator System object and block

The FIR Decimation block and the `dsp.FIRDecimator` System object now support variable-size input.

Min/Max logging instrumentation for float-to-fixed-point conversion of 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 (requires a Fixed-Point Designer license).

- `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.

HDL-optimized FFT and IFFT System objects and HDL-optimized Complex to Magnitude-Angle System object and block

This release introduces:

- `dsp.HDLFFT` and `dsp.HDLIFFT` System objects for the fast Fourier transform and inverse FFT, optimized for HDL code generation
- Complex to Magnitude-Angle HDL Optimized block and `dsp.ComplexToMagnitudeAngle` System object for converting complex inputs to magnitude and phase angle, optimized for HDL code generation using the CORDIC algorithm

Real input, bit-reversed output, reset input available on HDL-optimized FFT and IFFT

The following blocks and System objects now support real input, enable you to select or disable bit-reversed output, and provide an optional reset input:

- FFT HDL Optimized
- IFFT HDL Optimized
- `dsp.HDLFFT`
- `dsp.HDLIFFT`

Option to synthesize lookup table to ROM available on HDL-optimized FFT and IFFT blocks

To enable this feature, right-click the block, select **HDL Code > HDL Block Properties** and set **LUTRegisterResetType** to **none**.

The option to synthesize LUT to a ROM is not available on System objects.

Reduced latency of HDL-optimized FFT and IFFT

The FFT HDL Optimized and IFFT HDL Optimized blocks take fewer cycles to compute one frame of output than in previous releases. For instance, for the default 1024-point FFT, the latency in R2014a was 1589 cycles. In R2014b, the latency is 1148 cycles. The latency is displayed on the block icon.

Compatibility Considerations

If you have manually matched latency paths in models using the R2014a version of the FFT HDL Optimized and IFFT HDL Optimized block, rebalance those paths with the new latency.

CIC algorithm and HDL code generation for DC Blocker

This release adds an option to implement the DC Blocker using the CIC algorithm. You can generate HDL code from DC Blocker and `dsp.DCBlocker`. CIC mode is not yet supported for HDL code generation.

`dsp.FilterCascade` System object

This release introduces a new System object, `dsp.FilterCascade`, that constructs a cascade of filter System objects.

Phase Extractor block and `dsp.PhaseExtractor` System object

This release introduces a new block, Phase Extractor, and a new System object, `dsp.PhaseExtractor`, that extract the unwrapped phase from complex input signals.

Overrun and underrun reporting on audio device blocks and System objects

The following blocks and System objects now provide a count of samples lost to queue underrun/overrun since the last transfer of a frame to or from an audio device. You can use this information to debug throughput problems.

- To Audio Device

-
- From Audio Device
 - `dsp.AudioPlayer`
 - `dsp.AudioRecorder`

For an example of how to measure and tune audio throughput see [Measuring Audio Latency](#) example. You can open this example by typing `audiolatencymeasurement` at the MATLAB command line.

Unsigned input data type in `dsp.CICDecimator` and `dsp.CICInterpolator` System Objects

`dsp.CICDecimator` and `dsp.CICInterpolator` System objects now support unsigned input data type.

Logic Analyzer support for vector, enumerated, and complex inputs

The `dsp.logicAnalyzer` System object now supports vector, enumerated, and complex input signals.

System object support in Simulink For Each Subsystem

The new `supportsMultipleInstanceImpl` method enables the use of System objects in Simulink For Each Subsystem blocks. Include this method in your System object class definition file when you define a new kind of System object.

Getting Started Tutorials

This release adds 15 new tutorials, which illustrate a broad range of applications supported by the DSP System Toolbox software. There are new tutorials on the following topics:

- Streaming signal processing
- Filter design in MATLAB and Simulink
- Real-time audio processing and latency measurements
- Signal visualization in time and frequency
- Algorithm acceleration using code generation

- Multistage-multirate filtering for sample-rate conversion
- Authoring System objects
- Deploying MATLAB code and applications

See *Getting Started with DSP System Toolbox* for links to the new tutorials.

Functionality being removed or replaced for blocks and System objects

The Signal To Workspace block is now called To Workspace.

Certain functionality in the following blocks and System objects will be removed in future releases:

- Digital Filter block
- Variable Integer Delay block
- Delay block
- `dsp.DigitalFilter`
- `dsp.VariableIntegerDelay`
- `dsp.Delay`

These features will trigger a warning in R2014b. For most functionality, you can automatically update your model by running the Upgrade Advisor and selecting 'Check model for known block upgrade issues requiring compile time information'. See *Consult the Upgrade Advisor*.

Compatibility Considerations

Digital Filter and `dsp.DigitalFilter`

Use of the Digital Filter block and `dsp.DigitalFilter` System object in future releases is not recommended. Existing instances will continue to operate, but certain functionality will be disabled. If your model includes the functionality listed in the table below, you must update your model.

For future designs, choose from Discrete FIR Filter, Discrete Filter, Biquad Filter, or Allpole Filter blocks, and `dsp.FIRFilter`, `dsp.IIRFilter`, `dsp.BiquadFilter`, or `dsp.AllpoleFilter` System objects.

The functionality in the following table will be removed in future releases. The table describes the changes for the Digital Filter block. For `dsp.DigitalFilter`, apply the changes using the corresponding properties.

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Updating filter coefficients once per sample	Coefficient source is Input port(s), Input processing is Columns as channels (frame based), and Coefficient update rate is One filter per sample	Set Coefficient update rate to One filter per frame, insert the filter block in a For Iterator subsystem block, and use Variable Selector blocks to apply one set of coefficients to each input sample in a loop.	Yes
Column-based vector filter coefficients from input ports	Coefficient source is Input port(s)	Transpose your filter coefficients into a row vector by using a Transpose block.	Yes
Nonunity denominator coefficients from input ports	Coefficient source is Input port(s), Transfer function type is either IIR (poles & zeros) or IIR (all poles), and the First denominator coefficient = 1, remove a0 term in the structure check box is cleared.	Ensure the First denominator coefficient = 1, remove a0 term in the structure check box is selected, and scale your coefficients and initial values accordingly.	Yes

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Complex Biquad scale values	Transfer function type is IIR (poles and zeros) and the Filter structure is any SOS form	Use real scale values. Alternatively, for non-fixed-point input data types and zero initial conditions, set the scale value to 1, and add a Gain block at the filter's input port, where the gain value is equal to the product of the complex scale values.	Yes
State data type different from Accumulator data type	Transfer function type is FIR (all zeros) and the Filter structure is Direct form I transposed	Set the State data type to Same as accumulator.	No

Note To automatically apply the suggested fix in the Upgrade Advisor, select Check model for known block upgrade issues requiring compile time information.

Variable Integer Delay and `dsp.VariableIntegerDelay`

The DSP Variable Integer Delay block has been replaced with the Simulink Variable Integer Delay block. Existing instances of the DSP block will continue to operate, but certain functionality in the DSP block and `dsp.VariableIntegerDelay` System object will be disabled in future releases. If your model includes the functionality listed in the table below, you must update your model.

The functionality in the following table will be removed in future releases. The table describes the changes for the Variable Integer Delay block. For `dsp.VariableIntegerDelay`, apply the changes using the corresponding properties.

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Nonscalar delay	Using the block only. Nonscalar delay remains supported in the <code>dsp.VariableIntegerDelay</code> System object.	Insert your block in a For Each subsystem and partition the data and delay inputs to apply each delay value to the corresponding data channel.	Yes
Initial conditions specified as a vector	Input processing is Columns as channels (frame based), and the input has multiple channels (columns)	Specify the Initial conditions as a $1 \times NumChans \times R$ matrix, where <i>NumChans</i> is the number of input channels, and <i>R</i> is the maximum delay value.	Yes
	Input processing is Elements as channels (sample based), and the input has multiple channels (samples)	Specify the initial conditions as a $(dim1 \times dim2 \times \dots \times dimN) \times R$ array instead, where <i>dimM</i> is the <i>M</i> th input dimension and <i>R</i> is the maximum delay value.	Yes

Note To automatically apply the suggested fix in the Upgrade Advisor, select Check model for known block upgrade issues requiring compile time information.

Delay and `dsp.Delay`

The functionality in the following table will be removed in future releases. The table describes the changes for the Delay block. For `dsp.Delay`, apply the changes using the corresponding properties.

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Nonscalar delay	Using the block only. Nonscalar delay remains supported in <code>dsp.Delay System</code> object.	Use a Variable Integer Delay block in a For Each subsystem. Partition the data and delay inputs to apply each delay value to the corresponding data channel.	Yes
Delay specified in units of frames	Delay units is Frames	Set Delay units to Samples and set your new delay as the delay in frames multiplied by the frame length.	Yes

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
<p>Specification of different initial conditions for each channel but same conditions within a channel</p>	<p>The Specify different initial conditions for each channel check box is selected, and the Specify different initial conditions within a channel check box is cleared.</p>	<p>Clear both check boxes and specify the initial condition as a scalar. Alternatively, you can select the Specify different initial conditions within a channel check box and specify the initial conditions as follows:</p> <ul style="list-style-type: none"> • If Input processing is Columns as channels (frame based), specify the initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input channels. • If Input processing is Elements as channels (sample based), change input processing to Columns as channels (frame based), reshape the input to a row vector, and specify initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input elements. 	<p>Yes</p>

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Specification of different initial conditions within a channel but same conditions for each channel	The Specify different initial conditions within a channel check box is selected, and the Specify different initial conditions for each channel check box is cleared.	<p>Clear both check boxes and specify the initial condition as a scalar. Alternatively, you can select the Specify different initial conditions for each channel check box and specify the initial conditions as follows:</p> <ul style="list-style-type: none"> • If Input processing is Columns as channels (frame based), specify the initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input channels. • If Input processing is Elements as channels (sample based), set Input Processing to Columns as channels (frame based), reshape the input to a row vector, and specify initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input elements. 	Yes

Functionality	Applies When...	Use This Functionality Instead	Automatic fix using Upgrade Advisor?
Initial conditions specified as cell array		<p>Clear the Specify different initial conditions within a channel and Specify different initial conditions for each channel check boxes and specify the initial condition as a scalar. Alternatively, select both check boxes and modify the initial conditions as follows:</p> <ul style="list-style-type: none"> • If Input processing is Columns as channels (frame based), specify initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input channels. • If Input processing is Elements as channels (sample based), set Input Processing to Columns as channels (frame based), reshape the input to a row vector, and specify initial conditions as a <i>delay-by-NumChans</i> matrix, where <i>delay</i> is the delay value and <i>NumChans</i> is the number of input elements. 	Yes

Note To automatically apply the suggested fix in the Upgrade Advisor, select Check model for known block upgrade issues requiring compile time information.

Persistence mode in Vector Scope

The Vector Scope block no longer supports Persistence mode, which retained historical data on a single plot.

Compatibility Considerations

You do not need to update any existing model that had Persistence mode set. As of R2014b, you will not see historical data on your Vector Scope.

Code generation for additional DSP System Toolbox System objects

In R2014b, you can generate code from the following additional DSP System Toolbox System objects. Code generation from MATLAB code requires a MATLAB Coder license.

- `dsp.CICCompensationDecimator`
- `dsp.CICCompensationInterpolator`
- `dsp.FarrowRateConverter`
- `dsp.FilterCascade`

You cannot generate code directly from `dsp.FilterCascade`. Instead, first use the `dsp.FilterCascade.generateFilteringCode` method to generate a MATLAB function from the System object. Then generate C/C++ code from the MATLAB function.

- `dsp.FIRDecimator` for transposed structure
- `dsp.FIRHalfbandDecimator`
- `dsp.FIRHalfbandInterpolator`
- `dsp.PeakToPeak`
- `dsp.PeakToRMS`
- `dsp.PhaseExtractor`
- `dsp.SampleRateConverter`

-
- `dsp.StateLevels`

See System Objects in MATLAB Code Generation and Functions and System Objects Supported for C Code Generation.

Tunable amplitude on `dsp.SineWave`

The Amplitude property of `dsp.SineWave` is now tunable when the Method property is Differential or Trigonometric function.

R2014a

Version: 8.6

New Features

Compatibility Considerations

Up to four-times faster FIR filter simulation in MATLAB System object and Simulink block

This release introduces a refactoring of the Discrete FIR Filter block and `dsp.FIRFilter` System object to significantly improve simulation speed in multi-core processors. The refactored FIR simulation in MATLAB and Simulink leverages the Intel® Threading Building Blocks (TBB) library to optimize multi-core parallelism at the channel and frame level.

Optimized C code generation for ARM Cortex–M processors from System objects with MATLAB Coder and Embedded Coder

This release adds code-generation support for ARM Cortex-M processors in MATLAB for select System objects. With the supported System objects, you can generate C code that can be linked with the CMSIS library and compiled to provide an executable to run on ARM Cortex-M processors. To use the DSP System Toolbox Support Package for ARM Cortex-M Processors, you must have the following products in addition to the DSP System Toolbox: Simulink, Simulink Coder, Embedded Coder and MATLAB Coder. The following DSP System Toolbox System objects support the CMSIS library:

- `dsp.FIRFilter`
- `dsp.FIRDecimator`
- `dsp.FIRInterpolator`
- `dsp.LMSFilter`
- `dsp.BiquadFilter`
- `dsp.FFT`
- `dsp.IFFT`
- `dsp.Convolver`
- `dsp.CrossCorrelator`
- `dsp.Mean`
- `dsp.RMS`
- `dsp.StandardDeviation`
- `dsp.Variance`

Notch/peak filter and parametric equalizer filter System objects in MATLAB

This release introduces new second-order IIR notching/peaking and parametric equalizer filters. Use `dsp.NotchPeakFilter` to implement a peaking or notching filter. With `dsp.NotchPeakFilter`, you can control the center frequencies and 3-dB bandwidths of the peaks/notches with tunable properties.

Use `dsp.ParametricEQFilter` to implement a parametric equalizer with tunable gain, bandwidth, and center frequency.

Variable bandwidth FIR and IIR filter System objects in MATLAB

This releases introduces two new System objects, `dsp.VariableBandwidthFIRFilter` and `dsp.VariableBandwidthIIRFilter`, which allow you to vary the passband while filtering. `dsp.VariableBandwidthFIRFilter` and `dsp.VariableBandwidthIIRFilter` enable you to tune the filter in a computationally efficient way while preserving your filter structure.

Pink/Colored noise generation System object in MATLAB

This release introduces the ability to generate noise with a $1/f^\alpha$ power spectral density. You can set α equal to any value in the interval $[-2,2]$. Specifying $\alpha=1$ results in pink noise, while setting $\alpha=2$ produces Brownian noise. See `dsp.ColoredNoise` for details.

HDL optimized FFT and IFFT Simulink blocks

This release introduces FFT HDL Optimized and IFFT HDL Optimized blocks for the discrete Fourier transform (DFT) and inverse DFT optimized for HDL code generation.

Fixed-point data type support for FIR filter, in ARM Cortex-M support package

The Discrete FIR Filter from the Simulink workflow, and the `dsp.FIRFilter` from the MATLAB workflow, support fixed-point data types defined in the CMSIS library.

Choice of wrapping or truncating input of FFT, IFFT, and Magnitude FFT in MATLAB and Simulink

In the FFT, IFFT, and Magnitude FFT blocks, a boolean parameter has been added that is by default checked. This widget reads: **Wrap input data when FFT length is shorter than input length**, and it gives you the choice of wrapping or truncating the input, depending on the FFT length. If this parameter is checked, modulo-length data wrapping occurs before the FFT operation, given FFT length is shorter than the input length. If this property is unchecked, truncation of the input data to the FFT length occurs before the FFT operation.

In the `dsp.FFT` and `dsp.IFFT` System objects, a boolean property is added that is by default `true`. If this property is set to `true`, modulo-length data wrapping occurs before the FFT operation, given FFT length is shorter than the input length. If this property is set to `false`, truncation of the input data to the FFT length occurs before the FFT operation.

Variable-size input for biquad and LMS filters in MATLAB and Simulink

The Biquad Filter and LMS Filter blocks and the corresponding System objects, `dsp.BiquadFilter` and `dsp.LMSFilter`, now support variable-size input. In Simulink, this support means that the frame size (number of rows) can change during simulation. In a System object, this support allows the step method to handle an input that is changing in size.

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

In this release you can specify independent fixed-point data types for all `dsp.LMSFilter` System object fixed-point settings.

DC blocker System object and Simulink block

This release adds a new System object and Simulink block to remove the DC component of a signal. Use `dsp.DCBlocker` in MATLAB and the corresponding block, DC Blocker, in Simulink.

dsp.DigitalDownConverter and dsp.DigitalUpConverter now support C code generation

In this release you can generate C code for both digital down and up converters.

The isDone method of dsp.AudioFileReader honors PlayCount

The behavior of `isDone` has changed for this object. `isDone` returns `True` when EOF is reached `PlayCount` number of times. The default of `PlayCount` has changed from `Inf` to `1`.

Compatibility Considerations

If you had a while loop controlled by `isDone`, and you had set the `PlayCount` to `Inf`, you should now set it to the number of times you want the loop to be executed, otherwise you will have an infinite loop.

M4A replaced by MPEG4 in dsp.AudioFileWriter

In `dsp.AudioFileWriter`, the M4A file format has been changed to MPEG4.

Spectrogram cursors and CCDF plots in the spectrum analyzer

This release introduces cursors in `dsp.SpectrumAnalyzer` when the `SpectrumType` is set to `'Spectrogram'`. Additionally, you can now obtain complementary cumulative distribution function (CCDF) plots for your data.

Changed dsp.SpectrumAnalyzer property names

This table lists the `dsp.SpectrumAnalyzer` property name changes.

Old Property Name	New Property Name
Grid	ShowGrid
LegendSource	ShowLegend
MaxHoldTrace	PlotMaxHoldTrace
MinHoldTrace	PlotMinHoldTrace

Old Property Name	New Property Name
NormalTrace	PlotNormalTrace
TwoSidedSpectrum	PlotAsTwoSidedSpectrum

Compatibility Considerations

Update all instances of old names in your code to the new names.

Conversion to/from allpass from/to wave digital filter

This release introduces two new conversion functions, `allpass2wdf` and `wdf2allpass`, which enable you to convert from an allpass filter to a wave digital filter and from a wave digital filter to an allpass filter.

Transfer function estimation in Simulink

This release introduces a new Simulink block for transfer function estimation. You can find the Discrete Transfer Function Estimator block in the Power Spectrum Estimation library.

Updates to the Time Scope

The following updates have been made to the Time Scope.

- Array, structure, structure with time, and MAT-file logging formats
- Compact toolbar to allow more space for data display
- Scale X-axis, Y-axis, and XY-axes options, autoscaling and panning
- Default display does not show time-axis label. You can, optionally, turn the label on.
- Sampling option
- Model Configuration properties now honored by the Time Scope
- Snap-to-data cursors option to force cursors to data points
- Open Measurements panels saved when you save a model. Those panels reopen when you open the saved model.

Changed dsp.TimeScope property names

This table lists the dsp.TimeScope property name changes.

Old Property Name	New Property Name
Grid	ShowGrid
LegendSource	ShowLegend
MagnitudePhase	PlotAsMagnitudePhase
TimeSpanOverrunMode	TimeSpanOverrunAction

Compatibility Considerations

Update all instances of old names in your code to the new names.

Time Scope automatically switches to block-based sample time

The Time Scope uses port-based sample time, except in Simulink External or Rapid-Accelerator modes. In External and Rapid-Acceleration modes, the Time Scope switches to block-based sample time. Port-based sample time uses individual sample times for each input port. Block-based sample time uses the same sample time for the whole block.

dsp.LogicAnalyzer channel selection

In the dsp.LogicAnalyzer System object, you now can select individual or multiple channels. Then, you can find the next and previous transitions for the selected channel. Also, you can copy, paste, and move selected channels.

System object templates

The MATLAB New > System object menu now has three new class-definition file templates. The **Basic** template sets up a simple System object. The **Advanced** template includes additional features of System objects. The **Simulink Extension** template provides additional customization of the System object for use in the MATLAB System block.

System objects infer number of inputs and outputs from `stepImpl` method

When you create a new kind of System object that has a fixed number of inputs or outputs specified in the `stepImpl` method, you no longer need to include `getNumInputsImpl` or `getNumOutputsImpl` in your class definition file. The correct number of inputs and outputs are inferred from the `stepImpl` inputs and outputs, respectively.

System objects `setupImpl` method enhancement

When you create a new kind of System object and include the `setupImpl` method, you do not have to match the `setupImpl` method inputs to the `stepImpl` method inputs. If your `setupImpl` method does not use any input characteristics, such as, data type or size), you can include only the System object as the input argument.

System objects `infoImpl` method allows variable inputs

When you create a new kind of System object, you can use the `info` method to provide information specific to that object. The `infoImpl` method, which you include in your class-definition file, now allows `varargin` as an input argument.

System objects base class renamed to `matlab.System`

The System object base class, `matlab.system.System` has been renamed to `matlab.System`. If you use `matlab.system.System` when defining a new System object, an error message results.

Compatibility Considerations

Change all instances of `matlab.system.System` in your System objects code to `matlab.System`.

System objects Propagates mixin methods

Four new methods have been added to the Propagates mixin class. You use this mixin when creating a new kind of System object for use in the MATLAB System block in

Simulink. You use these methods to query the input and specify the output of a System object.

- `propagatedInputComplexity`
- `propagatedInputDataType`
- `propagatedInputFixedSize`
- `propagatedInputSize`

Code generation support for additional functions

This release introduces code generation support for the following functions. You must have the MATLAB Coder software to generate code.

- `ca2tf`
- `cl2tf`
- `firceqrip`
- `fireqint`
- `firgr`
- `firhalfband`
- `firminphase`
- `firnyquist`
- `firpr2chfb`
- `ifir`
- `iircomb`
- `iirgrpdelay`
- `iirlpnorm`
- `iirlpnormc`
- `iirnotch`
- `iirpeak`
- `tf2ca`
- `tf2cl`

R2013b

Version: 8.5

New Features

Compatibility Considerations

Support Package for ARM Cortex-M Processors

The DSP System Toolbox Support Package for ARM Cortex-M Processors allows you to model your signal processing algorithm in Simulink and generate C code. The generated code can be linked with the CMSIS library, and compiled to provide an executable to run on ARM Cortex-M processors. To use the DSP System Toolbox Support Package for ARM Cortex-M Processors, you must have the following products in addition to the DSP System Toolbox: Simulink, Simulink Coder, Embedded Coder, and MATLAB Coder. The following DSP System Toolbox blocks, which support this library, make it optimal for use in the ARM Cortex-M processors.

- Discrete FIR Filter
- FIR Decimation
- FIR Interpolation
- LMS Filter
- Biquad Filter
- FFT
- IFFT
- Correlation
- Convolution
- Mean
- RMS
- Variance
- Standard Deviation

To download and install this feature, select **Add-Ons > Get Hardware Support Packages** on the MATLAB Toolstrip. Then, use Support Package Installer to install the DSP System Toolbox Support Package for ARM Cortex-M Processors. For more information, see Support Package for ARM Cortex-M Processors.

Channel and distortion measurement, cursors, and spectrogram visualization using Spectrum Analyzer in MATLAB and Simulink

To enhance visualizing your data, channel measurements, distortion measurements, cursor measurements, and a spectrogram view have been added to the Spectrum Analyzer. Channel measurements show occupied bandwidth and adjacent channel power

ratio (ACPR) measurements. Distortion measurements show harmonic distortion and intermodulation distortion measurements. Cursor measurements show measurements between two cursors. A spectrogram is a display of the frequencies in a signal over time.

Channel mapping for multichannel audio devices in MATLAB and Simulink

The `dsp.AudioPlayer` and `dsp.AudioRecorder` System objects, and the To Audio Device and From Audio Device blocks, now support channel mapping. The term channel mapping is used to refer to a 1-1 mapping that associates channels on the selected audio device to channels of the data.

When you play audio, channel mapping allows you to specify on which channel of the audio device to output a specific channel of audio data. You can specify channel mapping as a vector of output channel indices corresponding to each output channel of data being written.

When you record audio, channel mapping allows you to specify on which channel of the audio data to input a specific channel of audio device. You can specify channel mapping as a vector of audio channel indices corresponding to each channel of data being read.

Variable-size support for FIR and Allpole filters in MATLAB and Simulink

In this release, `dsp.AllpoleFilter`, `dsp.FIRFilter`, Discrete FIR Filter block, and Allpole Filter block support variable-size input. Thus the number of rows (size of the frame) can vary.

Estimation of Power Spectrum, Cross Power Spectrum, and Transfer Function for streaming data in MATLAB

This release introduces `dsp.TransferFunctionEstimator`, `dsp.SpectrumEstimator`, and `dsp.CrossSpectrumEstimator`. The transfer function estimator, estimates the complex frequency-domain transfer function from time-domain data, based on the Welch averaged periodogram method. `dsp.TransferFunctionEstimator` provides functionality similar to the Signal Processing Toolbox function `tffestimate`, albeit in a streaming-friendly manner. The PSD and Cross-PSD estimators, are also provided using the Welch averaged periodogram

method, functionality similar to the Periodogram block in DSP System Toolbox. These three System objects support double- and single-precision floating point inputs. They also support C code generation.

Data logging and archiving using Time Scope in Simulink

Data logging and external mode data archiving have been added to the Time Scope block. You can now log scope data to a MAT-file.

MIDI control interface support in MATLAB

R2013b introduces these five functions that together provide the same functionality as that of the MIDI Controls block:

- `midiid` — Interactively identify a MIDI control.
- `midicontrols` — Open a group of MIDI controls for reading.
- `midiread` — Read the most recent values of the group of MIDI controls.
- `midisync` — Send values to update the group of MIDI controls.
- `midicallback` — Invoke a callback when an open control changes.

Among these functions, only `midicontrols`, `midiread`, and `midisync` support code generation.

Integer support on the output port of the MIDI Controls block

The MIDI Controls block now supports the `uint8` data type on the output port, for the range of 0 to 127, selecting the **raw MIDI** mode.

Kalman filter

This release introduces `dsp.KalmanFilter`. This filter supports C/C++ code generation, single- and double-precision floating point, MIMO, and optional control input. It also includes a subset of functionality in the corresponding block, Kalman Filter, including:

- Initial condition for estimated state
- Initial condition for estimated error covariance

-
- State transition matrix
 - Process noise covariance
 - Measurement matrix
 - Measurement matrix noise covariance
 - Output estimated measurement
 - Output estimated state
 - Multiple parallel filters
 - Disable update on a subset of filters

Adaptive filters using Lattice, Fast Transversal, Filtered-X LMS, and Frequency Domain algorithms in MATLAB

The following adaptive filters are introduced in this release:

- `dsp.AdaptiveLatticeFilter`
- `dsp.FastTransversalFilter` (no code generation)
- `dsp.FilteredXLMSFilter` (no code generation)
- `dsp.FrequencyDomainAdaptiveFilter`

They have the following features:

- C/C++ code generation support
- Floating-point data type support (double and single) for inputs
- Variable frame-size for inputs
- Real and complex inputs

Coupled allpass filter

This release introduces the `dsp.CoupledAllpass` filter, which implements IIR filters as the sum of two allpass filters operating in parallel. This filter allows you to use complex coefficients. It supports Variable-size input, floating-point filter analysis, and the filter coefficients are tunable. You can integrate this filter with filter design workflows such as `fdesign` and `filterbuilder`.

Functionality being removed or changed

Using `fdesign.pulseshaping` is discouraged because it is being removed in the future. However, it still runs when you try to use this functionality, but you are encouraged to use `rcosdesign` and `gaussdesign` instead.

Migrate away from `fdesign.pulseshaping`

Using `fdesign.pulseshaping` is not recommended. Use `rcosdesign` or `gaussdesign`. This table shows examples of how to replace the old function with the new functions.

<code>fdesign.pulseshaping</code>	<code>rcosdesign</code> and <code>gaussdesign</code>
<pre>sps=6; span=4; Beta=fdesign.pulseshaping(sps,... 'Square Root Raised Cosine',... 'Nsym,Beta',span,Beta); d1=design(f1); n1=d1.Numerator</pre>	<pre>n1n=rcosdesign(Beta,span,sps); n1n=n1n/max(n1n)*(-1/(pi*sps)... *(pi*(Beta-1)-4*Beta))</pre>
<pre>g1=fdesign.pulseshaping(sps,... 'Square Root Raised Cosine',... 'N,Beta',sps*span,Beta); h1=design(g1); k1=h1.Numerator</pre>	<pre>k1n=rcosdesign(Beta,span,sps); k1n=k1n/max(k1n)*(-1/(pi*sps)... *(pi*(Beta-1)-4*Beta))</pre>
<pre>f2=fdesign.pulseshaping(sps,... 'Raised Cosine,... 'Nsym,Beta',span,Beta); d2=design(f2); n2=d2.Numerator</pre>	<pre>n2n=rcosdesign(Beta,span,sps,'normal'); n2n=n2n/max(abs(n2n))/sps</pre>
<pre>g2=fdesign.pulseshaping(sps,... 'Raised Cosine',... 'N,Beta',sps*span,Beta); h2=design(g2); k2=h2.Numerator</pre>	<pre>k2n=rcosdesign(Beta,span,sps,'normal'); k2n=k2n/max(abs(k2n))/sps</pre>
<pre>BT=0.3; f3=fdesign.pulseshaping(sps,...'Gaussian',... 'Nsym,BT',span,BT); d3=design(f3); n3=d3.Numerator</pre>	<pre>n3n=gaussdesign(BT,span,sps)</pre>

Configuration dialog added to Logic Analyzer

A Visuals — Logic Analyzer Properties dialog box has been added to the `dsp.LogicAnalyzer`. This dialog box allows you to change the appearance of the scope. To open this dialog box, select **View > Configuration Properties**.

Complex trigger support in Time Scope

The Time Scope can plot signals as real/imaginary or magnitude/phase. In addition to using triggers from the real or imaginary data, you can now use magnitude or phase values from complex signals as triggers.

Default color changes for Array Plot, Time Scope, and Spectrum Analyzer

The following scopes use a new default color scheme to help emphasize the data being visualized:

- `dsp.ArrayPlot`
- `dsp.SpectrumAnalyzer` and Spectrum Analyzer block
- `dsp.TimeScope` and Time Scope block

MATLAB System Block to include System objects in Simulink models

The MATLAB System block is a new block in the Simulink User-Defined Functions library. Use this block to create a Simulink block that includes a System object™ in your model. This capability is useful for including your algorithm in your model.

Restrictions on modifying properties in System object Impl methods

When defining a new System object, certain restrictions affect your ability to modify a property.

You cannot use any of the following methods to modify the properties of an object:

- `cloneImpl`
- `getDiscreteStateImpl`

- `getDiscreteStateSpecificationImpl`
- `getNumInputsImpl`
- `getNumOutputsImpl`
- `getOutputDataTypeImpl`
- `getOutputSizeImpl`
- `isInputDirectFeedthroughImpl`
- `isOutputComplexImpl`
- `isOutputFixedSizeImpl`
- `validateInputsImpl`
- `validatePropertiesImpl`

This restriction is required by code generation, which assumes that these methods do not change any property values. These methods are validation and querying methods that are expected to be constant and should not impact the algorithm behavior.

Also, if either of the following conditions exist:

- You plan to generate code for the object
- The object will be used in the MATLAB System block

you cannot modify tunable properties for any of the following runtime methods:

- `outputImpl`
- `processTunedPropertiesImpl`
- `resetImpl`
- `setupImpl`
- `stepImpl`
- `updateImpl`

This restriction prevents tunable parameter updates within the object from interfering with updates from outside the generated code. Tunable parameters can only be changed from outside the generated code.

Compatibility Considerations

If any of your class definition files contain code that changes a property in one of the above `Impl` methods, move that property code into an allowable `Impl` method. Refer to the System object `Impl` method reference pages for more information.

System objects `matlab.system.System` warnings

The System object base class, `matlab.system.System`, has been replaced by `matlab.System`. If you use `matlab.system.System` when defining a new System object, a warning message results.

Compatibility Considerations

Change all instances of `matlab.system.System` in your System objects code to `matlab.System`.

Removing HDL Support for NCO Block

HDL support for the NCO block will be removed in a future release. Use the NCO HDL Optimized block instead.

Compatibility Considerations

In the current release, if you generate HDL code for the NCO block, a warning message appears. In a future release, any attempt to generate HDL code for the NCO block will cause an error.

R2013a

Version: 8.4

New Features

Compatibility Considerations

Allpass Filter System object

This release introduces an Allpass Filter System object™, `dsp.AllpassFilter`, which unifies functionality already existing in a number of `dfilt` objects under various names. `dsp.AllpassFilter` supports three different allpass structures and it can handle both single-section and cascaded configurations. It also supports double and single floating point, multichannel and variable-length input, tunability of filter coefficients and filter analysis.

Adaptive filter System objects using RLS and Affine Projection Filter

This release introduces two adaptive filter System objects, `dsp.RLSFilter` and `dsp.AffineProjectionFilter`. These System objects both support double and single floating point, and code generation.

Logic Analyzer System object

As of R2013a, DSP System Toolbox software provides a new Logic Analyzer System object that enables you to view the transitions of signals. To create a Logic Analyzer System object variable called *h*, at the MATLAB command prompt, type `h=dsp.LogicAnalyzer`. The Logic Analyzer has several graphical features:

- Multiple signals in a single window — The *y*-axis of the display can contain a number of channels, vertically tiled on top of each other.
- Ability to vary the display style — You can modify the name, height, color, and font size of each wave.
- Analog and Digital display formats — Both discrete and continuous signals can appear as waves or be tiled vertically in the display.
- Data numerical display options — You can display numerical values in various numeric systems, including unsigned decimal, signed decimal, hexadecimal, octal, and binary form. Such flexibility is especially useful for visualizing fixed-point signals.
- Cursors to mark points of interest and view values — By placing a cursor at a discrete time stamp, you can position a solid vertical line on the display and observe the values of each channel at that time stamp.
- Dividers to delineate groups of waves in a channel — By placing a divider on the display, you can separate waves by horizontal dashed lines.

For more information, see the `dsp.LogicAnalyzer` System object reference topic.

Audio System object support for tunability, variable frame size, variable number of channels, and writing MPEG-4 AAC

Effective 13a, variable frame size and variable number of channels are supported by `dsp.AudioPlayer`. If you use variable-size signals with this System object, you may experience sound dropouts when the size of the input changes. With the added support, you can avoid this behavior. Before you start the simulation, call `setup` for a signal with maximum dimensions.

This release also enhances `dsp.AudioPlayer` and `dsp.AudioRecorder` System objects to make sample rate, buffer size, and queue duration tunable. Tuning these properties stops and restarts the sound card, which creates a pause. The length of the pause depends on your buffer size and queue duration.

In addition, the `dsp.AudioFileWriter` System object and To Multimedia File block are enhanced to support MPEG-4 AAC audio files on Windows® 7, and Mac OS X. You can use both M4A and MP4 extensions. The following platform specific restrictions apply when writing these files:

Windows 7	Mac OS X
<ul style="list-style-type: none">• Only sample rates of 44100 and 48000 Hz are supported.	<ul style="list-style-type: none">• Only mono or stereo outputs are allowed.
<ul style="list-style-type: none">• Only mono or stereo outputs are allowed.	
<ul style="list-style-type: none">• Output data is padded on both front and back of the signal, with extra samples of silence. Windows AAC encoder places sharp fade-in and fade-out on an audio signal, causing the signal to be slightly longer in samples when written to disk.	<ul style="list-style-type: none">• Not all sampling rates are supported, although the Mac Audio Toolbox API do not explicitly specify a restriction.
<ul style="list-style-type: none">• A minimum of 1025 samples must be written to the MPEG-4 AAC file.	

Array Plot System object for displaying vectors or arrays in 2-D and Spectrum Analyzer block with enhanced controls and features such as peak finder

As of R2013a, DSP System Toolbox software provides a new Array Plot System object that enables you to visualize streaming data in two dimensions. Using Array Plot, you can visualize any set of data on the y -axis, opposite another set of data on the x -axis, labeling the x - and y -axes anything you choose. To create an Array Plot System object variable called h , at the MATLAB command prompt, type `h=dsp.ArrayPlot`. Array Plot contains the following panels.

- **Cursor Measurements** — shows cursors on all the displays. In the **Settings** pane, you can choose either waveform cursors, which are always attached to the signal data, or screen cursors, which may be placed anywhere on the axes. In the **Measurements** pane, you can see the x -axis value, y -axis value, and other calculated values at the locations of the cursors.
- **Signal Statistics** — displays the maximum, minimum, peak-to-peak difference, mean, median, and RMS values of a selected signal. It also shows the corresponding x -axis values at which the maximum and minimum values occur.
- **Peak Finder** — displays y -axis maxima and the corresponding x -axis values at which they occur. These displays allow you to modify the settings for peak threshold, maximum number of peaks, and peak excursion.

For more information, see the `dsp.ArrayPlot` System object reference topic.

As of R2013a, DSP System Toolbox software provides a new Spectrum Analyzer block to replace the Spectrum Scope block in the Sinks library. Using Spectrum Analyzer, you can view the power spectrum or power spectral density of signals. The graphical interface of the Spectrum Analyzer block resembles that of the `dsp.SpectrumAnalyzer` System object. Spectrum Analyzer contains the following panels.

- **Spectrum Settings** — enables you to modify settings to control how the spectrum is calculated. You can modify such parameters as frequency span, resolution bandwidth, number of spectral averages, and number of FFT points. You can also choose between a one-sided or two-sided spectrum and toggle normal, maximum hold, and minimum hold trace views.
- **Peak Finder** — displays spectral maxima and the corresponding frequencies at which they occur. These displays allow you to modify the settings for peak threshold, maximum number of peaks, and peak excursion.

You can programmatically modify parameters of the Spectrum Analyzer block using MATLAB code. To do so, you can first use the Simulink `get_param` function to get an instance of the `spbscopes.SpectrumAnalyzerConfiguration` class. Then, you can use dot notation or the `get` and `set` commands to modify properties of the Spectrum Analyzer block.

For more information, see the Spectrum Analyzer block reference topic.

Compatibility Considerations

All Simulink models containing Spectrum Scope blocks load with Spectrum Analyzer blocks in R2013a or later. The Spectrum Scope block had several dialog box parameters that do not appear in any Spectrum Analyzer block settings.

- Several options that were available on the Parameters dialog box of the Spectrum Scope block are no longer available or have changed. The parameters of Spectrum Scope map to Spectrum Analyzer parameters in the following manner.

R2012b Spectrum Scope Block Parameters dialog box Tab name	R2012b Spectrum Scope Parameter	R2013a Spectrum Analyzer Change	R2013a Spectrum Analyzer Equivalent Parameter
Scope Properties	Buffer input check box	R2013a Spectrum Analyzer does not require that input signals are buffered. Spectrum Analyzer determines the number of samples needed using the value of the RBW parameter. Regardless of whether the input is a frame-based or sample-based signal, Spectrum Analyzer calculates the spectrum once it has acquired the requisite number of samples.	<p>For Spectrum Scope blocks in R2012b or earlier models, the equivalent R2013a Spectrum Analyzer RBW value is given by the equation:</p> $RBW = \frac{K \times F_s}{L}$ <p>In the preceding equation, K is the window constant calculated for a segment length of 1000, F_s is the sample rate of the block, and L is the buffer length. If the input signal to the R2012b Spectrum Scope block was frame-based and the Buffer input check box was cleared, then the R2013a Spectrum Analyzer computes the RBW value with L set to the frame size of the input signal.</p>

R2012b Spectrum Scope Block Parameters dialog box Tab name	R2012b Spectrum Scope Parameter	R2013a Spectrum Analyzer Change	R2013a Spectrum Analyzer Equivalent Parameter
Scope Properties	Buffer size parameter	R2013a Spectrum Analyzer uses the RBW parameter to determine the requisite number of samples to calculate the spectrum, instead of using the buffer size or frame length.	For Spectrum Scope blocks in R2012b or earlier models, if the input signal was frame-based and the Buffer input check box was selected, then the R2013a Spectrum Analyzer computes the RBW value with L set to the value of the Buffer size parameter.
Scope Properties	Buffer Overlap parameter	R2013a Spectrum Analyzer has an Overlap % parameter that is directly related to buffer overlap.	R2013a Spectrum Analyzer will compute its Overlap % using the equation: $O_p = O_l / L \times 100$ <p>In the preceding equation, O_p is Overlap % parameter value, O_l is the R2012b Spectrum Scope Buffer overlap parameter value, and L is the buffer length.</p>
Scope Properties	Window parameter	R2013a Spectrum Analyzer does not have the Bartlett, Blackman, Triang, or Hanning settings.	Spectrum Scope blocks in R2012b or earlier models with a window parameter set to any of these values will have their Window parameter set to Hann in the R2013a Spectrum Analyzer.

R2012b Spectrum Scope Block Parameters dialog box Tab name	R2012b Spectrum Scope Parameter	R2013a Spectrum Analyzer Change	R2013a Spectrum Analyzer Equivalent Parameter
Scope Properties	Window Sampling parameter	R2013a Spectrum Analyzer does not have a Periodic option. All window sampling is now symmetric in the R2013a Spectrum Analyzer.	n/a
Display Properties	Persistence check box — this setting would execute the equivalent of the MATLAB hold on command, adding another line for each spectrum computation on the display.	This option is not available in the R2013a Spectrum Analyzer, which has replaced this feature with the trace options, Normal Trace , Max Hold Trace , and Min Hold Trace .	Spectrum Scope blocks in R2012b or earlier models with persistence enabled will have their Max Hold Trace check box selected in the R2013a Spectrum Analyzer.
Display Properties	Compact Display check box	There is no equivalent capability in the R2013a Spectrum Analyzer.	n/a

R2012b Spectrum Scope Block Parameters dialog box Tab name	R2012b Spectrum Scope Parameter	R2013a Spectrum Analyzer Change	R2013a Spectrum Analyzer Equivalent Parameter
Axis Properties	Inherit Sample time from input check box	R2013a Spectrum Analyzer always uses the sample time of the input signal.	n/a
Axis Properties	Frequency display limits parameter	R2013a Spectrum Analyzer determines the range of frequencies calculated based on the Full Span , FStart (Hz) , and FStop (Hz) parameters.	If this parameter was set to: <ul style="list-style-type: none"> • Auto — R2013a Spectrum Analyzer selects the Full Span check box on the Spectrum Settings panel, Main options pane. • User-defined — R2013a Spectrum Analyzer clears the Full Span check box on the Spectrum Settings panel Main options pane.
Axis Properties	Minimum frequency (Hz) parameter	R2013a Spectrum Analyzer determines the range of frequencies calculated based on the Full Span , FStart (Hz) , and FStop (Hz) parameters.	If the User-defined parameter was chosen, then this parameter maps to the R2013a Spectrum Analyzer FStart (Hz) parameter.
Axis Properties	Maximum frequency (Hz) parameter	R2013a Spectrum Analyzer determines the range of frequencies calculated based on the Full Span , FStart (Hz) , and FStop (Hz) parameters.	If the User-defined parameter was chosen, then this parameter maps to the R2013a Spectrum Analyzer FStop (Hz) parameter.

R2012b Spectrum Scope Block Parameters dialog box Tab name	R2012b Spectrum Scope Parameter	R2013a Spectrum Analyzer Change	R2013a Spectrum Analyzer Equivalent Parameter
Line Properties	Line visibilities , Line styles , Line markers , and Line colors parameters	There are no equivalent capabilities in the R2013a Spectrum Analyzer.	Once the simulation has started, you can modify the line styles, markers, and colors using the Style dialog box.

- The R2012b Spectrum Scope allowed you to retain the axes limits over multiple simulations by selecting **Axes > Save Axes Settings**. There is no equivalent capability in the R2013a Spectrum Analyzer. However, you can automatically scale the axes to a specified range using the Tools—Plot Navigation Properties dialog box.

Time Scope block with triggering and peak finder features

As of R2013a, DSP System Toolbox provides the following enhancements to the Time Scope block and the `dsp.TimeScope` System object:

- “Triggers Panel” on page 10-11
- “Peak Finder Features” on page 10-11
- “Panning Capability” on page 10-11
- “Programmatic Access” on page 10-11
- “Scale Axes Limits After 10 Updates” on page 10-12

For more information on these enhancements, see the Time Scope block reference topic or the `dsp.TimeScope` System object reference topic.

Triggers Panel

The Time Scope contains a new **Triggers** panel that allows you to pause the display only when certain events occur. You can use the Triggers panel when you want to align or search for events.


- In the **Mode** pane, you choose how often the display should update.
- In the **Source / Type** pane, you choose the type of events on which to stop. Events include edges (rising, falling, or both), pulse width, transition, runt, window, or timeout.
- In the **Levels / Timing** pane, you can set the trigger level and hysteresis value.
- In the **Delay / Holdoff** pane, you can offset the trigger position by a fixed delay or set the minimum possible time between trigger events.

To access the **Triggers** panel, in the Time Scope toolbar, click the Triggers button (). Alternatively, in the Time Scope menu, select **Tools > Triggers**.

Peak Finder Features

Effective in R2013a, the peak finder now has the option of displaying more than 10 maxima at once. You can also use the Peaks panel to toggle the labels for each local maximum. This capability allows you to choose whether all the labels show time, value, or both time and value. R2013a also provides the capability to sort in either ascending or descending order by value or by time.

Panning Capability

Effective in R2013a, you can pan in all directions on the Time Scope display. In the Time Scope toolbar, click the Pan button (). Alternatively, in the Time Scope menu, select **Tools > Pan**. Then, click and drag the mouse to the left or right to view a different range of data on the *time*-axis. Click and drag the mouse up or down to see a different Amplitude range on the *y*-axis.

Programmatic Access

Effective in R2013a, you can change the properties of the Time Scope block using MATLAB commands. To do so, you can first use the Simulink `get_param` function to get an instance of the `Simulink.scopes.TimeScopeConfiguration` class. Then, you can use dot notation or the `get` and `set` commands to modify properties of the Time Scope block.

For more information on these enhancements, see the `Simulink.scopes.TimeScopeConfiguration` class reference topic.

Scale Axes Limits After 10 Updates

Effective in R2013a, you can scale the axes limits of the Time Scope displays soon after the simulation starts. To do so, in the Time Scope menu, select **Tools > Scale Axes Limits After 10 Updates**.

Change of the default for audio hardware API on Linux

In this release, the default for audio hardware API on Linux® has changed from OSS to ALSA.

Change of the default for audio file formats in multimedia blocks and audio file reader and writer System objects

In the To Multimedia File block and the `dsp.AudioFileWriter` System object, the file format default is now WAV. In the From Multimedia File block and the `dsp.AudioFileReader` System object, the file format default is now MP3.

Change of property default in the audio file reader System object

In the `dsp.AudioFileReader` System object, the property `OutputDataType` returns doubles as default.

Removal of the signalblks package

The `signalblks` package has been removed. Instead, for System object classes, and properties, use the `dsp` package.

Compatibility Considerations

To automatically update the existing code, where the `signalblks` package is used, run `sysobjupdate`. This function updates System object code to work in the current release. The application recursively searches the specified folder and subfolders for MATLAB files that contain renamed System object packages, classes, and properties.

Scope Snapshot display of additional scopes in Simulink Report Generator

Using Simulink Report Generator™ software, you can include snapshots of the display produced by a Scope block in a generated report. The Scope Snapshot component, which inserts images of the Simulink Scope block and XY Graph block, now supports the Time Scope block and Spectrum Analyzer block in DSP System Toolbox software.

Note This feature requires that you have a license for the Simulink Report Generator product.

For more information, see the Simulink Report Generator product documentation.

Unoriented vector treated as column vector in the Biquad Filter

Starting this release, the unoriented vector is treated as a column vector in numerator and denominator coefficient ports.

NCO HDL Optimized block

The NCO HDL Optimized block provides hardware friendly control signals, optional reset port, and an optional external dither input port. It also provides a reset function that resets the phase to its initial value during the sinusoid output generation. In addition, it includes an option to output the internal phase and hardware-friendly control signals including valid in and valid out.

HDLNCO System object

The `dsp.HDLNCO` System object, like the `dsp.NCO` System object, generates real or complex sinusoidal signals. In addition, the NCO HDL-optimized System object provides hardware friendly control signals, optional reset signal, and an optional external dither input signal.

HDL code generation for NCO HDL Optimized block and System object

Release R2013a provides HDL code generation support for the new NCO HDL Optimized block and dsp.HDLNCO System object. To generate HDL code, you must have an HDL Coder license.

Support for nonpersistent System objects

You can now generate code for local variables that contain references to System objects. In previous releases, you could not generate code for these objects unless they were assigned to persistent variables.

New method for action when System object input size changes

The new `processInputSizeChangeImpl` method allows you to specify actions to take when an input to a System object you defined changes size. If an input changes size after the first call to `step`, the actions defined in `processInputSizeChangeImpl` occur when `step` is next called on that object.

Scaled double data type support for System objects

System objects now support scaled double data types.

R2012b

Version: 8.3

New Features

Compatibility Considerations

SpectrumAnalyzer System object

As of R2012b, DSP System Toolbox software provides a new Spectrum Analyzer System object that enables you to view the power spectrum or power spectral density of signals. To create a Spectrum Analyzer System object variable called *h*, at the MATLAB command prompt, type `h=dsp.SpectrumAnalyzer`. The Spectrum Analyzer has several panels and dialog boxes that allow you to perform the following operations:

- **Spectrum Settings panel** — The **Spectrum Settings** panel enables you to modify settings to control the manner in which the spectrum is calculated. You can modify such parameters as frequency span, resolution bandwidth, number of spectral averages, and number of FFT points. You can also choose between a one-sided or two-sided spectrum and toggle normal, maximum hold, and minimum hold trace views. To hide or display the **Spectrum Settings** panel, in the Spectrum Analyzer toolbar, select the Spectrum Settings button () . Alternatively, in the Spectrum Analyzer menu, select **View > Spectrum Settings**.
- **Peak Finder panel** — The **Peak Finder** panel displays maxima and the frequencies at which they occur. These displays allow you to modify the settings for peak threshold, maximum number of peaks, and peak excursion. In the Spectrum Analyzer toolbar, click the Peak Finder button () . Alternatively, in the Spectrum Analyzer menu, select **Tools > Measurements > Peak Finder**.
- **Properties dialog box** — The Spectrum Analyzer provides a dialog box that allows you to control the most common properties of a display, including the grid lines, titles, *y*-axis labels and *y*-axis limits. To change options for a display, in the Spectrum Analyzer toolbar, click the Properties button () . Alternately, in the Spectrum Analyzer menu, select **View > Properties**. You can also right-click the display, and select **Properties**.
- **Style dialog box** — The Spectrum Analyzer allows you to customize the style of displays using a Style dialog box. You can change the color of the figure containing the displays, the background and foreground colors of display axes, and properties of lines in a display. To view or modify the line style of the active signal, in the Spectrum Analyzer menu, select **View > Style**. You can also right-click the display and select **Style**.
- **Axes Scaling Options** — The Spectrum Analyzer enables you with the ability to automatically scale the axes to a specified range. In the Spectrum Analyzer menu, select **Tools > Axes Scaling Options** to modify these settings. To manually scale the axes to the limits you specified, click the Scale Axes Limits button () .

For more information, see the `dsp.SpectrumAnalyzer System` object reference topic.

Cross-platform support for reading and writing WAV, FLAC, OGG, MP3 (read only), MP4 (read only), and M4a (read only)

The `dsp.AudioFileReader System` object, and the `From Multimedia File` block, now support the following audio file formats on all platforms:

- MP3
- MP4
- M4a
- WAV
- FLAC
- OGG

The `dsp.AudioFileWriter System` object, and the `To Multimedia File` block, now support the following audio file formats on all platforms:

- WAV
- FLAC
- OGG

Support for code generation for CICDecimator and CICInterpolator System objects

The following `System` objects now support code generation in MATLAB via the `codegen` command:

- `dsp.CICDecimator`
- `dsp.CICInterpolator`

To use the `codegen` function, you must have a MATLAB Coder license. See `Use System Objects in MATLAB Code Generation` for more information.

Support for HDL code generation for multichannel Discrete FIR Filter block

Discrete FIR Filter block accepts vector input and supports multichannel implementation for better resource utilization.

- With vector input and channel sharing option `on`, the block supports multichannel fully parallel FIR, including direct form FIR, sym/antisym FIR, and FIRT. Support for all implementation parameters, for example: multiplier pipeline, add pipeline registers.
- With vector input and channel sharing option `off`, the block instantiates one filter implementation for each channel. If the input vector size is N , N identical filters are instantiated.

For fully parallel architecture option for FIR filters only.


Time Scope enhancements, including new cursors, embedded simulation controls, and External and Rapid Accelerator modes

As of R2012b, DSP System Toolbox provides the following enhancements to the Time Scope block and the `dsp.TimeScope` System object:

- “Cursor measurements panel” on page 11-4
- “Additional embedded simulation controls” on page 11-5
- “Support for external mode and rapid accelerator mode” on page 11-5
- “Properties dialog box” on page 11-6
- “Axes Maximization” on page 11-7
- “Automatic calculation of Time Span” on page 11-7
- “ReduceUpdates property” on page 11-7
- “Support for conditional subsystems” on page 11-8

Cursor measurements panel


The Time Scope contains a new **Cursor Measurements** panel that shows cursors on all the Time Scope displays. In the **Settings** pane, you may choose either waveform cursors, which are always attached to the signal data, or screen cursors, which may be placed anywhere on the axes. The **Measurements** pane shows the time, amplitude, and other

calculated values at the locations of the cursors. In the Time Scope toolbar, click the Cursor Measurements button (). Alternatively, in the Time Scope menu, select **Tools > Measurements > Cursor Measurements**.

For more information, see the Time Scope block reference topic or the `dsp.TimeScope` System object reference topic.

Additional embedded simulation controls

Effective in R2012b, additional embedded simulation controls are available through the Simulation Toolbar and the Simulation Stepping Options dialog box. In previous releases, the Time Scope block featured a Simulation Toolbar. With this toolbar, you could control the progression of increasing simulation time from the Time Scope GUI by clicking the Run, Pause, Stop, and Next Step buttons. In R2012b, the Simulation Stepping Options dialog box provides you with the ability to further control the simulation behavior. This dialog box allows you to enable the button on the Simulation Toolbar to take a Previous Step. Additionally, you can pause the simulation at a specified time, specify previous stepping options, and modify the number of steps for forward and backward movement. To access these controls, from the Time Scope menu, select **Simulation > Stepping Options**. Alternatively, if previous stepping is disabled, in the Time Scope toolbar, click the Previous Step button. The Simulation Stepping Options dialog box appears.

You can enable Time Scope to show a Previous Step button (), which allows you to move the simulation time backward by one time step. In the Simulation Stepping Options dialog box, in the **Previous stepping options** group, select the **Enable Previous Stepping** check box. To test this feature, first run the simulation, and then pause the simulation. When the simulation is paused, you can now click the Previous Step button to regress the simulation back by one time step.

Note This feature is available for the Time Scope block but not for the `dsp.TimeScope` System object.

For more information, see the Time Scope block reference topic.

Support for external mode and rapid accelerator mode

As of R2012b, the Time Scope block supports two additional simulation modes in Simulink, External mode and Rapid Accelerator mode. You can use External mode to

tune block parameters in real time and view block outputs in many types of blocks and subsystems. External mode establishes communication between a host system, where the Simulink environment resides, and a target system, where the executable runs after it is generated by the code generation and build process. For more information about External mode, see *Host/Target Communication* in the Simulink Coder product documentation.


You can use Rapid Accelerator mode as a method to increase the execution speed of your Simulink model. Rapid Accelerator mode creates an executable that includes the solver and model methods. This executable resides outside of MATLAB and Simulink. Rapid Accelerator mode uses External mode to communicate with Simulink. For more information about Rapid Accelerator mode, see *Acceleration* in the Simulink product documentation.

Note This feature is available for the Time Scope block but not for the `dsp.TimeScope` System object.

For more information about Time Scope, see the Time Scope block reference topic.

Properties dialog box

As of R2012b, the Time Scope block provides a centralized location where you can modify the most important properties of a display. The Properties dialog box contains the most frequently modified Time Scope settings, including all the parameters from the Tools:Plot Navigation Options dialog box and the Visuals:Time Domain Options dialog box. It also includes **Open at Start of Simulation** and **Number of Input Ports** from the **File** menu. To open this dialog box, in the Time Scope toolbar, click the Properties

button (). Alternately, in the Time Scope menu, select **View > Properties**. You can also right-click on the display and select **Properties**.

Note This feature is available for the Time Scope block but not for the `dsp.TimeScope` System object. In the `dsp.TimeScope` System object GUI, when you select **View > Properties**, the Visuals:Time Domain Options dialog box appears, as in R2012a. This same dialog box also appears when you right-click on the display and select **Properties**.

For more information about Time Scope, see the Time Scope block reference topic.

Axes Maximization

In R2012b, you can specify whether to display the Time Scope block or System object in maximized axes mode. In this mode, the axes are expanded to fill the entire display. In each display, there is no space to show titles or axis labels. The values at the axis tick marks appear on top of the axes. You can select one of the following options:

- `Auto` — In this mode, the axes appear maximized in all displays only if the **Title** and **Y-Axis label** parameters are empty for every display. If you enter any value in any display for either of these parameters, the axes are not maximized.
- `On` — In this mode, the axes appear maximized in all displays. Any values entered into the **Title** and **Y-Axis label** parameters are hidden.
- `Off` — In this mode, none of the axes appear maximized.

The default setting is `Auto`. In the Time Scope GUI, you can set this property in the **Main** pane of the Properties dialog box. To change options using the `dsp.TimeScope` System object, set the `MaximizeAxes` property to the intended option. For more information, see the Time Scope block reference topic or the `dsp.TimeScope` System object reference topic.

Automatic calculation of Time Span

As of R2012b, the Time Scope block can automatically calculate the **Time Span** parameter using the simulation **Start Time** and **Stop Time** parameters. By default, the Time Scope block has the **Time Span** parameter set to **Auto calculate**. To modify the **Time Span** parameter to use a different value, open the Properties dialog box and click the **Time** tab.

Note This feature is available for the Time Scope block but not for the `dsp.TimeScope` System object.

For more information about Time Scope, see the Time Scope block reference topic.

ReduceUpdates property

As of R2012b, the Time Scope System object has an additional property called `ReduceUpdates`. By default, this property is set to `true`. When this property is `true`, the Time Scope updates the displays at a rate not exceeding 20 hertz. When you set this property to `false`, the Time Scope updates every time the `step` method is called. The

simulation speed is faster when this property is set to `true`. Using this property is equivalent to selecting the **Reduce Updates to Improve Performance** check box in the **Simulation** menu of the Time Scope GUI.

For more information about this property, see the `dsp.TimeScope System` object reference topic.

Support for conditional subsystems

In previous releases, the Time Scope block could be used within an enabled subsystem. As of R2012b, the Time Scope block can also be placed in a triggered subsystem, an enabled and triggered subsystem, and a function-call subsystem. For more information about these types of subsystems, see Conditional Subsystems in the Simulink documentation.

Note This feature is available for the Time Scope block but not for the `dsp.TimeScope System` object.

For more information about Time Scope, see the Time Scope block reference topic.

Source and sink blocks being replaced

The following Windows platform blocks now map to other existing blocks that work on all platforms:

Deprecated blocks	Blocks mapped to
From Wave Device	From Audio Device
To Wave Device	To Audio Device
From Wave File	From Multimedia File
To Wave File	To Multimedia File

This mapping is transparent; no `slupdate` is needed. When you open an existing model that contains the original blocks, the replacement blocks are automatically substituted. If you save the model, the replacement blocks are saved instead of the original blocks.

Compatibility Considerations

Because mapped blocks do not have identical functionality, incompatibilities can be introduced in certain cases. The From Wave File can have an optional `start-of-file`

indicator port, whereas the From Multimedia File cannot. Therefore, if you load an old model where From Wave File has the `start-of-file` port, a broken link results. In this case, a warning message appears, providing a link to `start_of_file_example`, which shows you how to correct the problem.

Discrete IIRFilter and AllpoleFilter System objects

This release introduces a discrete IIR Filter System object `dsp.IIRFilter` and a discrete Allpole Filter System object `dsp.AllpoleFilter`.

The IIR Filter System object implements the algorithm, inputs, and outputs described on the Discrete Filter block reference page. The object properties correspond to the block parameters. Both this object and its corresponding block let you specify whether to process inputs as individual samples or as frames of data. This System object supports code generation.

The Allpole Filter System object implements the algorithm, inputs, and outputs described on the Allpole Filter block reference page. The object properties correspond to the block parameters. Both this object and its corresponding block let you specify whether to process inputs as individual samples or as frames of data. This System object supports code generation.

Support for MATLAB Compiler for CICDecimator and CICInterpolator System objects

The following System objects are now supported by MATLAB Compiler™:

- `dsp.CICDecimator`
- `dsp.CICInterpolator`

For more information, see [Using System Objects with MATLAB Compiler](#).

Code generation support for SignalSource System object

`dsp.SignalSource` now support code generation in MATLAB via the `codegen` command. To use the `codegen` function, you must have a MATLAB Coder license. See [Use System Objects in MATLAB Code Generation](#) for more information.

Behavior change of locked System objects for loading, saving, and cloning

In the previous release, saving, loading, and cloning a locked System object would result in an unlocked System object. This System object had the same property values as the one from which it was cloned, but not the same internal state.

In this release, it does not matter whether you save a locked System object into a MAT file and load it later or clone a locked System object using the `clone` method. In either case, the result is a locked System object with the same property values and the same internal states.

There are, however, a few exceptions. For the following System objects, if you call the `clone` method, the resulting System object is not locked, but if you save or load the System object into and from a MAT file, the result is a locked System object.

- `dsp.MatFileWriter`
- `dsp.AudioRecorder`
- `dsp.AudioPlayer`

Thus, for the above System objects, if you call the `clone` method, you get an unlocked System object with the same property values.

Another exception involves the following System objects:

- `dsp.AudioFileWriter`
- `dsp.AudioFileReader`

For these System objects, if you save a locked System object to a MAT file and load it later, you get an unlocked System object with the same property values. However you do not get the same internal states. This behavior is the same as in the previous release.

Behavior change of statistics blocks for variable-size inputs

When the inputs are of variable size, the running mode behavior of the following blocks has changed:

- Mean
- RMS

-
- Variance
 - Standard Deviation
 - Minimum
 - Maximum

If the **Input processing** parameter is set to `Elements as channels` (sample based), the block state is reset when any input dimension changes.

If the **Input processing** parameter is set to `Columns as channels` (frame based), then the behavior depends on two options:

- When the number of input channels (i.e., number of columns) changes, the block state is reset to its initial condition.
- When the number of input channels remains the same, there is no reset, even if the channel length (i.e., number of rows) changes.

Simulation state save and restore for additional blocks

In this release, there are additional blocks that support simulation state save and restore. These are:

- Cumulative Sum
- Cumulative Product
- Mean
- Variance
- RMS
- Standard Deviation
- Queue
- Stack

Note The Queue and Stack blocks do not support SimState save and restore in dynamic memory allocation mode.

For more information on simulation state save and restore, see [Save and Restore Simulation State as SimState](#).

For Each subsystem support for additional blocks

In this release, most DSP blocks have been updated to support the For Each subsystem. For details about the For Each subsystem, see For Each. For a list of supported blocks, see the *For Each Subsystem Support* column in the table titled *Simulink Block Data Type Support for DSP System Toolbox*. This table can be accessed by typing `showsignalblockdatatypetable` at the command line.

Multi-instance model referencing support for additional blocks

In this release, most DSP blocks have been updated to support multi-instance normal mode model referencing. For details about model referencing, see Model Reference. The blocks that support model referencing are the same blocks that support the For Each subsystem. Therefore for a list of supported blocks, see the *For Each Subsystem Support* column in the table titled *Simulink Block Data Type Support for DSP System Toolbox*. This table can be accessed by typing `showsignalblockdatatypetable` at the command line.

Expanded analysis support for filter System objects

In the previous release, the filter analysis methods for `dfilt` and `mfilt` objects were extended to filter System objects. This release expands the number of supporting analysis methods to include the following:

- `noisepsd`
- `noisepsdopts`
- `freqrespest`
- `freqrespopts`

For a comprehensive list of supported analysis methods, see Analysis Methods for Filter System Objects.

Removal of the `signalblks` package

In this release, the `signalblks` package is being removed and any instantiation of a `signalblks` object causes an error message. In future releases, the functionality will be

removed entirely, so you should now use the corresponding object in the DSP System Toolbox.

Discrete filter block visible in DSP library

In addition to accessing the Discrete Filter block from the Simulink library, you can now also access it from the DSP System Toolbox library.

System object tunable parameter support in code generation

You can change tunable properties in user-defined System objects at any time, regardless of whether the object is locked. For System objects predefined in the software, the object must be locked. In previous releases, you could tune System object properties only for a limited number of predefined System objects in generated code.

save and load methods for System objects

You can use the `save` method to save System objects to a MAT file. If the object is locked, its state information is saved, also. You can recall and use those saved objects with the `load` method.

You can also create your own `save` and `load` methods for a System object you create. To do so, use the `saveObjectImpl` and `loadObjectImpl`, respectively, in your class definition file.

Save and restore SimState not supported for System objects

The Save and Restore Simulation State as SimState option is no longer supported for any System object in a MATLAB Function block. This option was removed because it prevented parameter tunability for System objects, which is important in code generation.

Compatibility Considerations

If you need to save and restore simulation states, you may be able to use a corresponding Simulink block, instead of a System object.

Map integer delay to RAM on Delay block

`UseRAM` is a block-level parameter on the `IntegerDelay` block that you access with the HDL Block properties GUI. You assign it an `On` or `Off` value:

- `On`: Map the integer delay to a RAM. `On` is not a guarantee that a RAM is inferred: if all conditions are met (including the threshold criteria), only then is the RAM inferred.
- `Off`: The integer delay is always mapped to registers.

HDL support for System objects

HDL support for the following System objects has been added with release R2012b:

- `dsp.BiquadFilter`

HDL resource sharing for Biquad Filter block

HDL support for second-order section, direct-form I and second-order section, direct-form II filter structures has been added with release R2012b.

Supported architectures:

- Fully Parallel ('default' interface)

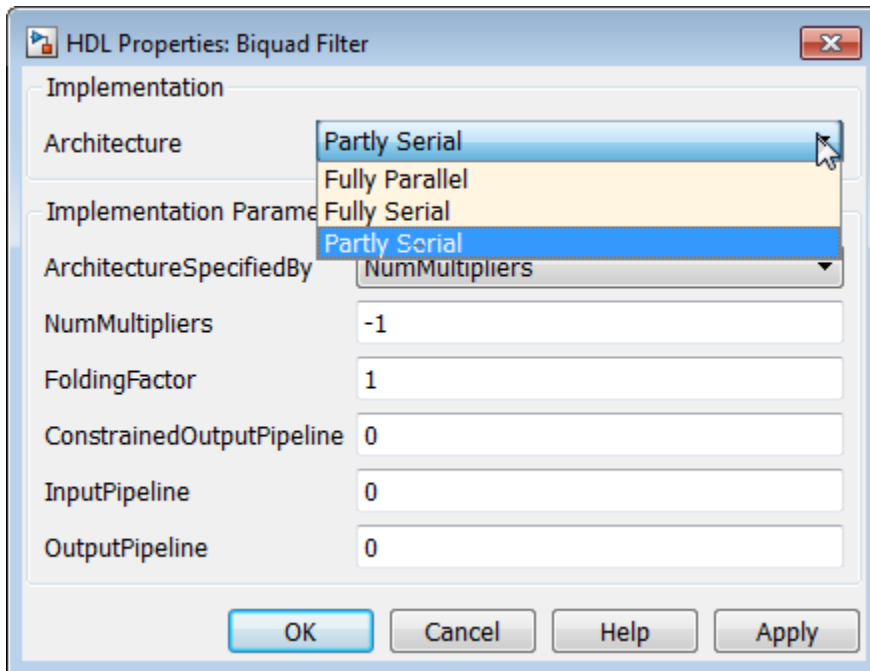
`AddPipelineRegisters`, `ConstrainedOutputPipeline`, `CoeffMultipliers`, `InputPipeline`, `OutputPipeline`

- Fully Serial

`ConstrainedOutputPipeline`, `InputPipeline`, `OutputPipeline`

- Partly Serial

`ConstrainedOutputPipeline`, `InputPipeline`, `OutputPipeline`, `ArchitectureSpecifiedBy`, `FoldingFactor`, `NumMultipliers`



R2012a

Version: 8.2

New Features

Bug Fixes

Compatibility Considerations

Frame-Based Processing

Beginning in R2010b, MathWorks has been significantly changing the handling of frame-based processing. For more information, see “Frame-Based Processing” on page 13-2 in the R2011b Release Notes.

The following sections provide more detailed information about the specific R2012a DSP System Toolbox software changes that are helping to enable the transition to the new paradigm for frame-based processing:

- “Inherited Option of the Input Processing Parameter Now Warns” on page 12-2
- “Logging Frame-Based Signals in Simulink” on page 12-3
- “Model Reference and Using slupdate” on page 12-4
- “Removing Mixed Frameness Support for Bus Signals on Unit Delay and Delay” on page 12-4
- “Audio Output Sampling Mode Added to the From Multimedia File Block” on page 12-5

Inherited Option of the Input Processing Parameter Now Warns

Some DSP System Toolbox blocks are able to process both sample- and frame-based signals. After the transition to the new way of handling frame-based processing, signals will no longer carry information about their frame status. Blocks that can perform both sample- and frame-based processing will have a new parameter that allows you to specify the appropriate processing behavior.

To prepare for this change, many blocks received a new **Input processing** parameter in previous releases. You can set this parameter to `Columns as channels` (frame based) or `Elements as channels` (sample based), depending upon the type of processing you want. The third choice, `Inherited` (this choice will be removed - see release notes), is a temporary selection that is available to help you migrate your existing models from the old paradigm of frame-based processing to the new paradigm.

In this release your model will warn when the following conditions are all met for any block in your model:

- The **Input processing** parameter is set to `Inherited` (this choice will be removed - see release notes).

- The input signal is sample based.
- The input signal is a vector, matrix, or N-dimensional array.

Compatibility Considerations

To eliminate this warning, you must upgrade your existing models using the `slupdate` function. The function detects all blocks that have `Inherited` (this choice will be removed - see release notes) selected for the **Input processing** parameter. It then asks you whether you would like to upgrade each block. If you select yes, the function detects the status of the frame bit on the input port of the block. If the frame bit is 1 (frames), the function sets the **Input processing** parameter to `Columns as channels` (frame based). If the bit is 0 (samples), the function sets the parameter to `Elements as channels` (sample based).

In a future release, the `frame bit` and the `Inherited` (this choice will be removed - see release notes) option will be removed. At that time, the **Input processing** parameter in models that have not been upgraded will automatically be set to either `Columns as channels` (frame based) or `Elements as channels` (sample based). The option set will depend on the library default setting for each block. If the library default setting does not match the parameter setting in your model, your model will produce unexpected results. Additionally, after the frame bit is removed, you will no longer be able to upgrade your models using the `slupdate` function. Therefore, you should upgrade your existing modes using `slupdate` as soon as possible.

Logging Frame-Based Signals in Simulink

In this release, a new warning message appears when a Simulink model is logging frame-based signals and the **Signal logging format** is set to `ModelDataLogs`. In `ModelDataLogs` mode, signals are logged differently depending on the status of the frame bit, as shown in the following table:

Status of Frame Bit	Today	When Frame Bit Is Removed
Sample-based	3-D array with samples in time in the third dimension	3-D array with samples in time in the third dimension
Frame-based	2-D array with frames in time concatenated in the first dimension	3-D array with samples in time in the third dimension

This warning advises you to switch your **Signal logging format** to `Dataset`. The `Dataset` logging mode logs all 2-D signals as 3-D arrays, so its behavior is not dependent on the status of the frame bit.

When you get the warning message, to continue logging signals as a 2-D array:

- 1 Select **Simulation > Model Configuration Parameters > Data Import/Export**, and change **Signal logging format** to `Dataset`. To do so for multiple models, click on the link provided in the warning message.
- 2 Simulate the model.
- 3 Use the `dsp.util.getLogArray` function to extract the logged signal as a 2-D array.

Model Reference and Using `slupdate`

In this release, the Model block has been updated so that its operation does not depend on the frame status of its input signals.

Compatibility Considerations

In a future release, signals will not have a `frameness` attribute, therefore models that use the Model block must be updated to retain their behavior. If you are using a model with a Model block in it, follow the steps below to update your model:

- 1 For both the child and the parent models:
 - In the **Model Configuration Parameters** dialog box, select the **Diagnostics > Compatibility** pane.
 - Change the **Block behavior depends on input frame status** parameter to `warning`.
- 2 For both the child and the parent models, run `slupdate`.
- 3 For the child model only:
 - In the **Model Configuration Parameters** dialog box, select the **Diagnostics > Compatibility** pane.
 - Change the **Block behavior depends on input frame status** parameter to `error`.

Removing Mixed Frameness Support for Bus Signals on Unit Delay and Delay

This release phases out support for buses with mixed sample- and frame-based elements on Simulink's Unit Delay and Delay blocks. Support is also removed from the DSP

System Toolbox's Delay block. When the frame bit is removed in a future release, any Delay block that has a bus input of mixed framedness will produce different results.

Compatibility Considerations

This incompatibility is phased over multiple releases. In 2012a, the blocks will produce a warning message. In a future release, when the frame bit is removed, the blocks will produce an error message.

Audio Output Sampling Mode Added to the From Multimedia File Block

The From Multimedia File block now provides a new parameter. This parameter allows you to select frame- or sample-based audio output processing.

System Object Enhancements

- “Code Generation for System Objects” on page 12-5
- “New MAT-File Reader and Writer System Objects” on page 12-5
- “New System Object Option on File Menu” on page 12-6
- “Variable-Size Input Support for System Objects” on page 12-6
- “Data Type Support for System Objects” on page 12-6
- “New Property Attribute to Define States” on page 12-6
- “New Methods to Validate Properties and Get States from System Objects” on page 12-6
- “matlab.system.System changed to matlab.System” on page 12-6

Code Generation for System Objects

System objects defined by users now support C code generation. To generate code, you must have the MATLAB Coder product.

New MAT-File Reader and Writer System Objects

R2012a adds two new System objects, `dsp.MatFileReader` and `dsp.MatFileWriter`. These System objects stream data into and out of MAT-files.

New System Object Option on File Menu

The File menu on the MATLAB desktop now includes a **New > System object** menu item. This option opens a System object class template, which you can use to define a System object class.

Variable-Size Input Support for System Objects

System objects that you define now support inputs that change size at run time.

Data Type Support for System Objects

System objects that you define now support all MATLAB data types as inputs and outputs.

New Property Attribute to Define States

R2012a adds the new `DiscreteState` attribute for properties in your System object class definition file. Discrete states are values calculated during one step of an object's algorithm that are needed during future steps.

New Methods to Validate Properties and Get States from System Objects

The following methods have been added:

- `validateProperties` – Checks that the System object is in a valid configuration. This applies only to objects that have a defined `validatePropertiesImpl` method.
- `getDiscreteState` – Returns a struct containing System object properties that have the `DiscreteState` attribute.

`matlab.system.System` changed to `matlab.System`

The base System object class name has changed from `matlab.system.System` to `matlab.System`.

Compatibility Considerations

The previous `matlab.system.System` class will remain valid for existing System objects. When you define new System objects, your class file should inherit from the `matlab.System` class.

Time Scope Enhancements

- “Time Domain Measurements in Time Scope” on page 12-7
- “Multiple Display Support in Time Scope” on page 12-7
- “Style Dialog Box in Time Scope” on page 12-8
- “Sampled Data as Stairs in Time Scope” on page 12-8
- “Complex Data Support in Time Scope” on page 12-9
- “Additional Time Scope Enhancements” on page 12-9

Time Domain Measurements in Time Scope

As of R2012a, the Time Scope block and System object support the time domain signal measurements **Signal statistics**, **Bilevel measurements**, and **Peak finder**. The **Signal Statistics** panel displays the maximum, minimum, peak-to-peak difference, mean, median, and RMS values of a selected signal. It also displays the times at which the maximum and minimum values occur. The **Bilevel Measurements** panel displays information about a selected signal’s transitions, overshoots or undershoots, and cycles. The **Peak Finder** panel displays maxima and the times at which they occur. These displays allow you to modify the settings for peak threshold, maximum number of peaks, and peak excursion.

To use the new time domain measurements features in the Time Scope block, click one of the three corresponding buttons in the Time Scope toolbar. You can also access these panels by selecting **Measurements** from the **Tools** menu.

See the Time Scope reference topic for more information.

Multiple Display Support in Time Scope

R2012a allows you to choose to have multiple displays in the Time Scope, using both the block and the System object. This feature allows you to tile your screen into a number of separate displays, up to a grid of 4 rows and 4 columns. You may find multiple displays useful when the Time Scope takes multiple input signals.

To set the number of displays on the Time Scope, click the layout button in the Time Scope toolbar. You can also select **Layout** from the **View** menu. To set the number of displays using the `dsp.TimeScope` System object, set the `LayoutDimensions` property.

To change options for a display, select **Properties** from the **View** menu. Select the **Display** tab, and use the menu to select the display you want to update. You can also

right-click on the axes, and select **Properties**. To change options using the `dsp.TimeScope` System object, set the `ActiveDisplay` property to the intended display number.

See the Time Scope reference topic for more information.

Style Dialog Box in Time Scope

R2012a enhances the Time Scope block and System object by allowing you to customize the style of displays using a Style dialog box. You are able to change the color of the figure containing the displays, the background and foreground colors of display axes, and properties of lines in a display.

The Style dialog box replaces the **Line Properties** menu item that was used in previous releases for customizing line properties. To open the Style dialog box, select **Style** from the **View** menu.

Note This release changes the Time Scope default axes and line colors. The Time Scope initially displays the axes as black instead of white, as shown in previous releases. For a real, single-channel signal, Time Scope now displays a yellow line instead of a blue line. Models containing Time Scope blocks that were created using older versions of DSP System Toolbox will not be affected by this change.

See the Time Scope reference topic for more information.

Sampled Data as Stairs in Time Scope

In previous releases, the Time Scope plotted a sampled signal as lines connecting each of the sampled values. This approach is similar to the functionality of the MATLAB `line` or `plot` function. In R2012a, the Time Scope block and System object can also plot a sampled signal as horizontal lines. These lines represent a sample value for a discrete sample period connected by vertical lines to represent a change in values occurring at each new sample. This type of plot is commonly called a Stairstep graph and has functionality similar to that of the MATLAB `stairs` function. Stairstep graphs are useful for drawing time history graphs of digitally sampled data.

To display a sampled signal as a Stairstep graph, first select **Style** from the **View** menu. The Style dialog box opens, allowing you to set the **Plot type** drop down box to **Stairs**. The three options available are **Line**, **Stairs**, and **Auto**. If using the `dsp.TimeScope` System object, set the `PlotType` property to the string `'stairs'`.

See the Time Scope reference topic for more information.

Complex Data Support in Time Scope

Beginning in this release, the Time Scope block and System object will support complex data input. The complex data is displayed by default in real and imaginary form as differently colored lines on the same axes. Alternately, you can display the magnitude and phase of the signal on separate axes in the same display.

To change the complex data options in the Time Scope display, select **Properties** from the **View** menu. Then, select or deselect the **Plot signals as magnitude and phase** check box. You can also right-click on the axes and select **Properties**. To change these properties using the `dsp.TimeScope` System object, set the `MagnitudePhase` property to either `true` or `false`.

See the Time Scope reference topic for more information.

Additional Time Scope Enhancements

In addition to the modifications mentioned above, R2012a also includes the following enhancements to the Time Scope:

Ability to Change the Time Units of the Display

In previous releases Time Scope always displayed time in metric units. In R2012a, the Time Scope block and System object allow you to label the X-axis in two additional ways. First, you can ensure that the X-axis is always labeled as `Time (seconds)` and that the appropriate power of 10 appears in the bottom-right corner of the Time Scope display. Second, you can remove the units in the X-axis label entirely.

To change the manner in which the time units are displayed, select **Properties** from the **View** menu. Then, set the **Time Units** parameter to either `Seconds` or `None`. The default option is `Metric` (based on `Time Span`). To change these properties using the `dsp.TimeScope` System object, set the `TimeUnits` property to either `Seconds` or `None`. The default property value is `Metric`.

See the Time Scope reference topic for more information.

Simulink Enumerations Supported in Time Scope Block

In previous releases, the Time Scope block supported input signals of floating-point and fixed-point data types. R2012a adds support for Simulink enumerated data types.

Note This feature is available only for the Time Scope block but not for the Time Scope System object. Also, support is provided only for Simulink enumerations, but not for generic MATLAB enumeration classes.

See the Time Scope reference topic for more information.

ASIO Support in To/From Audio Device Blocks and Objects

The To and From Audio Device blocks and the `dsp.AudioPlayer` and `dsp.AudioRecorder` system objects all now support ASIO as an API. ASIO is used to communicate with the audio hardware. To set ASIO as the Audio Hardware API, select **Preferences** from the MATLAB Toolstrip. Then select DSP System Toolbox from the tree menu. If the ASIO selection is disabled, it is due to the ASIO device not being connected.

Video Processing Enabled for the DSP System Toolbox Multimedia File Blocks

The To Multimedia File and From Multimedia File blocks no longer require a Computer Vision System Toolbox license for video processing. You can use the DSP From Multimedia File block to read video and the To Multimedia File block to write video files.

System Objects Integrated into Filter Design Workflow

- “Integration of System Objects into Filter Design via `fdesign`, `FDATool`, and `Filterbuilder`” on page 12-10
- “Convert `dfilt` and `mfilt` Filter Objects to System Objects” on page 12-11
- “Filter Analysis and Conversion Methods for System Object Filters” on page 12-11

Integration of System Objects into Filter Design via `fdesign`, `FDATool`, and `Filterbuilder`

In R2012a, you can use the `fdesign` workflow and interactive tools, `fdatool` and `filterbuilder`, to create IIR, FIR, and multirate System object filters.

Using the interactive tools, you can generate MATLAB code to construct a System object filter. You can also generate MATLAB code to filter data with your System object that is compatible with C/C++ code generation. C/C++ code generation requires the MATLAB Coder software.

Convert `dfilt` and `mfilt` Filter Objects to System Objects

In R2012a, you can convert `dfilt` and `mfilt` objects to System objects. Use the `sysobj` method to convert an existing `dfilt` or `mfilt` object to a System object. Refer to the command-line help for `dfilt.sysobj` and `mfilt.sysobj` for details on supported single-rate and multirate filter structures.

Filter Analysis and Conversion Methods for System Object Filters

The R2012a release extends filter analysis and conversion methods for `dfilt` and `mfilt` objects to System object filters. For System object filters, you can examine the filter magnitude, impulse, step, zero phase, group delay, and phase responses. You can also view a pole-zero plot of your filter's z -transform.

Additionally, you can obtain detailed measurements and implementation costs for your System object filter. For System object filters, you can determine if the phase response is linear. For FIR linear-phase filters, you can determine the type of linear phase. You can also assess the stability of your filter design and whether your design represents a minimum-phase or maximum-phase system.

To obtain a comprehensive list of supported methods and links to the command-line help, enter

```
dsp.SystemObjectFilter.helpFilterAnalysis
```

at the command line, where `SystemObjectFilter` is a specific System object filter class name. For example:

```
dsp.BiquadFilter.helpFilterAnalysis
```

New Measurement Workflow

- “Measurements for Bilevel Pulse Waveforms” on page 12-11
- “System Objects for Peak-to-RMS and Peak-to-Peak Measurements” on page 12-12

Measurements for Bilevel Pulse Waveforms

The R2012a release introduces System objects that perform a number of basic measurements on bilevel pulse waveforms. These measurements include:

- State-level estimation for bilevel pulse waveforms using the histogram method. See the help for `dsp.StateLevels` for details.

- Transition metrics for bilevel pulse waveforms. The `System` object, `dsp.TransitionMetrics`, determines low-, middle-, and high-reference level crossings and also duration and slew rate. You can also use `dsp.TransitionMetrics` to measure the behavior of bilevel waveforms in pretransition and posttransition regions such as overshoot, undershoot, and settling time.

Using `dsp.PulseMetrics`, you can measure transition rise and fall times. `dsp.PulseMetrics` contains a superset of the capabilities found in `dsp.TransitionMetrics`.

- Cycle metrics for bilevel pulse waveforms. You can use `dsp.PulseMetrics` to measure pulse width, pulse separation, pulse period, and duty cycle.

System Objects for Peak-to-RMS and Peak-to-Peak Measurements

The R2012a release introduces `System` objects to measure the root-mean-square (RMS) value of a waveform. These `System` objects also measure the peak-to-RMS and peak-to-peak values. For details, see the reference pages for `dsp.RMS`, `dsp.PeakToRMS`, and `dsp.PeakToPeak`.

Discrete FIR Filter System Object

This release introduces a discrete FIR Filter `System` object, which filters each channel of the input using static or time-varying FIR filter implementations. This `System` object implements the algorithm, inputs, and outputs described on the Discrete FIR Filter block reference page. The object properties correspond to the block parameters. Both this object and its corresponding block let you specify whether to process inputs as individual samples or as frames of data. The object uses the `FrameBasedProcessing` property. The block uses the **Input processing** parameter. This `System` object supports code generation.

Inverse Dirichlet Sinc-Shaped Passband Design Added to Constrained FIR Equiripple Filter

The R2012a release adds the ability to design a constrained FIR equiripple filter with an inverse-Dirichlet-sinc-shaped passband using `firceqrip`. An inverse-Dirichlet-sinc-shaped response is often used to compensate for the Dirichlet-sinc-shaped response of a cascade integrator comb (CIC) filter. An inverse-sinc-shaped response is a valid approximation to the response of a CIC filter only when the sampling rate change is

sufficiently high. The Dirichlet sinc provides a more exact match to the response of a CIC filter.

Code Generation Support Added to FIR Decimator System Object

In this release, FIR Decimator has been added to the list of objects that can now support code generation in MATLAB via the `codegen` function.

Filter Block Enhancements

- “IC/Coefficient Parameter Ports in the Simulink Discrete Filter and Discrete Transfer Function” on page 12-13
- “Reset Port for Resetting Filter State in Filter Blocks” on page 12-13

IC/Coefficient Parameter Ports in the Simulink Discrete Filter and Discrete Transfer Function

The Simulink Discrete Filter and Discrete Transfer Function blocks now have the capability of specifying the numerator and denominator coefficients via either input parameter ports or block dialog boxes. Also, the initial filter states can be specified via an input parameter port or a block dialog box.

Reset Port for Resetting Filter State in Filter Blocks

The Simulink Discrete Filter and Discrete Transfer Function blocks now allow the filter states to be reset via a reset parameter port called External reset.

Discrete FIR Filter Block Coefficient Port Changes

In this release, if you feed a column vector input into the coefficient port of the Discrete FIR Filter block, the block issues a command-line warning. This warning will state that column vector inputs to the coefficient port are not supported and that you will see an error message in future releases.

Compatibility Considerations

You are advised to run `slupdate` to insert a reshape block and transpose the input from a column vector to a row vector.

Statistics Blocks and Objects Warning for Region of Interest Processing

ROI processing will be removed in a future release. Currently, ROI processing is available only if you have a Computer Vision System Toolbox license. If you do not have a license for that product, you can still use ROI processing, but you are limited to the use of ROI type rectangles.

Compatibility Considerations

When you use region of interest (ROI) processing, MATLAB will issue a warning. ROI processing will be removed in a future release.

New and Updated Demos

R2012a adds the following new demo:

Using System Objects with MATLAB Coder — Shows you how to use the MATLAB Coder product to generate code for a MATLAB file that uses System objects.

Additionally, this release updates the Arbitrary Magnitude Filter Design demo to include examples that showcase the new capabilities of the arbitrary magnitude filter design.

R2011b

Version: 8.1

New Features

Bug Fixes

Compatibility Considerations

Frame-Based Processing

In signal processing applications, you often need to process sequential samples of data at once as a group, rather than one sample at a time. DSP System Toolbox documentation refers to the former as frame-based processing and the latter as sample-based processing. A frame is a collection of samples of data, sequential in time.

Historically, Simulink-family products that can perform frame-based processing propagate frame-based signals throughout a model. The frame status is an attribute of the signals in a model, just as data type and dimensions are attributes of a signal. The Simulink engine propagates the frame attribute of a signal by means of a frame bit, which can either be on or off. When the frame bit is on, Simulink interprets the signal as frame based and displays it as a double line, rather than the single line sample-based signal.

General Product-Wide Changes

Beginning in R2010b, MathWorks® started to significantly change the handling of frame-based processing. In the future, frame status will no longer be a signal attribute. Instead, individual blocks will control whether they treat inputs as frames of data or as samples of data. To learn how a particular block handles its input, you can refer to the block reference page.

To make the transition to the new paradigm of frame-based processing, many blocks have received new parameters. You can view an example of how to use these parameters to control sample- and frame-based processing in R2011b and future releases. To open the model, type `ex_inputprocessing` at the MATLAB command line. This model demonstrates how a block can process a signal as sample based or frame based, depending on the setting of that block's **Input processing** parameter.

Notice that when the Discrete FIR Filter and Time Scope blocks are configured to perform frame-based processing, they interpret columns as channels and treat the 2-by-2 input signal as two independent channels. Conversely, when the blocks are configured to perform sample-based processing, they interpret elements as channels and treat the 2-by-2 input signal as four independent channels. For further information about sample- and frame-based processing, see [Sample- and Frame-Based Concepts](#).

The following sections provide more detailed information about the specific R2011b DSP System Toolbox software changes that are helping to enable the transition to the new way of frame-based processing:

-
- “Logging Signals in Simulink” on page 13-4
 - “Triggered to Workspace” on page 13-4
 - “Digital Filter Design Block” on page 13-5
 - “Filterbuilder, FDATool and the Filter Realization Wizard Block” on page 13-6
 - “Changes to Row Vector Processing for dsp.Convolver, dsp.CrossCorrelator, and dsp.Interpolator System Objects” on page 13-6

Compatibility Considerations

During this transition to the new way of handling frame-based processing, both the old way (frame status as an attribute of a signal) and the new way (each block controls whether to treat inputs as samples or as frames) will coexist for a few releases. For now, the frame bit will still flow throughout a model, and you will still see double signal lines in your existing models that perform frame-based processing.

- **Backward Compatibility** — By default, when you load an existing model in R2011b any new parameters related to the frame-based processing change will be set to their backward-compatible option. For example, if any blocks in your existing models received a new **Input processing** parameter this release, that parameter will be set to `Inherited` (this choice will be removed - see release notes) when you load your model in R2011b. This setting enables your existing models to continue working as expected until you upgrade them. Because the inherited option will be removed in a future release, you should upgrade your existing models as soon as possible.
- **slupdate Function** — To upgrade your existing models to the new way of handling frame-based processing, you can use the `slupdate` function. Your model must be compilable in order to run the `slupdate` function. The function detects all blocks in your model that are in need of updating, and asks you whether you would like to upgrade each block. If you select yes, the `slupdate` function updates your blocks accordingly.
- **Timely Update to Avoid Unexpected Results** — It is important to update your existing models as soon as possible because the frame bit will be removed in a future release. At that time, any blocks that have not yet been upgraded to work with the new paradigm of frame-based processing will automatically transition to perform their library default behavior. The library default behavior of the block might not produce the results you expected, thus causing undesired results in your models. Once the frame bit is removed, you will no longer be able to upgrade your models using the

`slupdate` function. Therefore, you should upgrade your existing modes using `slupdate` as soon as possible.

Logging Signals in Simulink

R2011b adds new capabilities to the DSP System Toolbox product for logging signals in Simulink. When you log signals using the `Dataset` logging mode, you can now use DSP System Toolbox utility functions to help you access that logged data in either a 2-D or 3-D format. For more information about selecting a signal logging format, see [Specifying the Signal Logging Data Format](#) in the Simulink documentation.

After you log a signal using the `Dataset` logging mode, you can choose to extract that logged signal in either a 2-D or 3-D format. To fully support this new workflow, the following utility functions and class have been added to the DSP System Toolbox product:

- `dsp.util.getLogArray` — Formats and returns a 2-D or 3-D MATLAB array from a logged signal in a `Dataset` object.
- `dsp.util.getSignalPath` — Returns all paths to signals with a specified name in the `Dataset` object.
- `dsp.util.SignalPath` — Contains path information for signals in `Simulink.SimulationData.Dataset` objects.

Triggered to Workspace

R2011b adds a new **Save 2-D signals as** parameter to the Triggered to Workspace block. This parameter allows you to specify whether the block saves 2-D signals as 2-D arrays or as 3-D arrays. To provide for backward compatibility, the **Save 2-D signals as** parameter also has an option `Inherit from input` (this choice will be removed – see release notes). When you select this option, the block saves sample-based data as a 3-D array and frame-based data as a 2-D array.

Compatibility Considerations

In a future release, the following option will be removed: `Inherit from input` (this choice will be removed – see release notes). From this time forward, you must specify whether the block saves 2-D signals as 2-D or 3-D arrays. The block will no longer make that choice based on the status of the frame bit.

You can use the `slupdate` function to upgrade your existing models that contain a Triggered To Workspace block. The function detects whether your models contain any

Triggered To Workspace blocks with the **Save 2-D signals as** parameter set to `Inherit from input` (this choice will be removed – see release notes). If you do, the function detects the status of the frame bit and sets the **Save 2-D signals as** parameter accordingly.

- If the input signal is frame based, the function sets the **Save 2-D signals as** parameter to 2-D array (concatenate along first dimension).
- If the input signal is sample based, the function sets the **Save 2-D signals as** parameter to 3-D array (concatenate along third dimension).

Digital Filter Design Block

R2011b adds a new **Input processing** parameter to the Digital Filter Design block. This parameter allows you to choose whether you want the block to perform sample- or frame-based processing on the input. You can set this parameter to either `Elements as channels` (sample based) or `Columns as channels` (frame based). The third choice, `Inherited` (this choice will be removed – see release notes), is a temporary selection. This additional option will help you as you move control of frame-based processing from the signals to the blocks themselves.

Compatibility Considerations

When you load an existing model R2011b, all Digital Filter Design blocks in your model will have the new **Input processing** parameter. By default, it will be set to `Inherited` (this choice will be removed – see release notes). This setting enables your existing models to continue to work as expected until you upgrade them. Although your old models will still work when you open and run them in R2011b, you should upgrade them as soon as possible.

You can upgrade your existing models using the `slupdate` function. The function detects all blocks that have `Inherited` (this choice will be removed – see release notes) selected for the **Input processing** parameter. It then asks you whether you would like to upgrade each block. If you select yes, the function detects the status of the frame bit on the input port of the block. If the frame bit is 1 (frames), the function sets the **Input processing** parameter to `Columns as channels` (frame based). If the bit is 0 (samples), the function sets the parameter to `Elements as channels` (sample based).

In a future release, the frame bit and the `Inherited` (this choice will be removed – see release notes) option will be removed. At that time, the **Input**

processing parameter on blocks in models that have not been upgraded will automatically be set to the block's library default setting. In the case of the Digital Filter Design block, the library default setting is `Columns as channels (frame based)`. If the library default setting does not match the parameter setting in your model, your model will produce unexpected results.

Additionally, after the frame bit is removed, you will no longer be able to upgrade your models using the `supdate` function. Therefore, you should upgrade your existing modes using `supdate` as soon as possible.

Filterbuilder, FDATool and the Filter Realization Wizard Block

R2011b adds new **Input processing** and **Rate options** parameters to `filterbuilder`, `FDATool` and the Filter Realization Wizard block. For `filterbuilder`, these new parameters are available when you click the **Generate Model** button on the Code Generation pane of the dialog box. For `FDATool` and the Filter Realization Wizard block, the new parameters are available on the Realize Model pane of the dialog box. When you use the **Realize Model** button to create a block, you can use the **Input processing** parameter to specify whether the block will perform sample- or frame-based processing on its input.

If you are creating a multirate filter block, the **Rate options** parameter will also be available. This parameter allows you to specify whether the filter block you create will `Enforce single-rate processing` or `Allow multirate processing`.

Changes to Row Vector Processing for `dsp.Convolver`, `dsp.CrossCorrelator`, and `dsp.Interpolator System` Objects

In previous releases, the `dsp.Convolver`, `dsp.CrossCorrelator`, and `dsp.Interpolator System` objects processed row vector inputs as a column vector. As of R2011b, these objects now process row vector inputs as a row vector (multiple channels).

Compatibility Considerations

Starting in R2011b, you must update your code to transpose the row vector data to a column vector before providing it as an input to the `dsp.Convolver`, `dsp.CrossCorrelator`, or `dsp.Interpolator System` objects.

Custom System Objects

You can now create custom System objects in MATLAB. This capability allows you to define your own System objects for time-based and data-driven algorithms, I/O, and visualizations. The System object API provides a set of implementation and service methods that you incorporate into your code to implement your algorithm. See Define New System Objects in the DSP System Toolbox documentation for more information.

New Allpole Filter Block

R2011b adds a new Allpole Filter block to the Filtering/Filter Implementations library. This block provides direct form, direct form transposed, and Lattice AR allpole filter structures.

New Audio Weighting Filter Functionality

R2011b adds new audio weighting filter functionality to MATLAB and Simulink. In MATLAB, you can now design audio weighting filters in the `filterbuilder` GUI or by using the preexisting `fdesign.audioweighting` object. In Simulink, you can use the new Audio Weighting Filter block from the Filtering/Filter Designs library.

Time Scope Enhancements

R2011b includes the following enhancements to the Time Scope:

- **Improvements to default signal names in the scope legend** — In previous releases, the default names for signals displayed by the Time Scope block were Channel 1, Channel 2, Channel 3, etc. In R2011b, the default naming convention has been improved to also identify the source of the signal. For example, if the input to the Time Scope block is two separate two channel signals named SignalA and SignalB, the default legend names would appear as:

```
SignalA:1, SignalA:2, SignalB:1, SignalB:2
```

See the Time Scope reference topic for more information.

- **New scrolling display mode simplifies debugging process** — R2011b adds a new option to the Time Scope block and System object. This option allows you to specify how the scope displays new data beyond the visible time span. In previous releases, the scope always displayed new data up until it reached the maximum X-

axis limit. When the data reached the maximum X-axis limit of the scope window, the scope cleared the display and updated the time offset value. It then displayed subsequent data points starting from the minimum X-axis limit. In the new scrolling display mode, the scope scrolls old data to the left to make room for new data on the right side of the scope display. This mode is graphically intensive and can affect run-time performance, but it is beneficial for debugging and for monitoring time-varying signals.

To use the new scrolling display mode in the Time Scope block, set the **Time span overrun mode** parameter to `Scroll` on the **Visuals:TimeDomainOptions** dialog box. To use the new scrolling display mode in the `dsp.TimeScope` System object, set the `TimeSpanOverrunMode` property to `Scroll`. By default, both the block and the System object display data using the previously supported `Wrap` mode.

New Arbitrary Group Delay Design Support

This release adds a new `fdesign.arbgrpdelay` filter specification object. Arbitrary group delay filters are allpass filters useful for correcting phase distortion introduced by other filters. Systems with nonlinear phase responses result in nonconstant group delay, which causes dispersion of the frequency components of the signal. This type of phase distortion can be undesirable even if the magnitude distortion introduced by the filter produces the desired effect. In these cases, you can compensate for the phase distortion by cascading the frequency-selective filter with an allpass filter that compensates for the group delay.

Arbitrary Magnitude Responses Now Support Minimum Order and Minimum/Maximum Phase Equiripple Design Options

R2011b adds new minimum order, minimum phase equiripple, and maximum phase equiripple design options. These design options are now available on the Arbitrary Response design panel in `filterbuilder` and through the `fdesign.arbmag` filter specification object.

Support for Constrained Band Equiripple Designs in MATLAB and Simulink

R2011b adds support for constrained band equiripple designs to the following filter response types:

fdesign Filter Specification Object	filterbuilder Response String	dspfdesign Library Block
<code>fdesign.arbmag</code>	<code>arbmag</code>	Arbitrary Response Filter
<code>fdesign.bandpass</code>	<code>bandpass</code>	Bandpass Filter
<code>fdesign.bandstop</code>	<code>bandstop</code>	Bandstop Filter
<code>fdesign.differentiator</code>	<code>diff</code>	Differentiator Filter

New Sinc Frequency Factor and Sinc Power Design Options for Inverse Sinc Filters

R2011b adds two new design options for designing inverse sinc filters in MATLAB and Simulink. The new design options allow you to control the sinc frequency factor and sinc power for inverse sinc filters designed with `fdesign.isinclp`, the new `fdesign.isinchp`, the `isinc` filterbuilder response type, or the Inverse Sinc Filter block in the `dspfdesign` library.

These new design options allow you to design inverse sinc lowpass filters with a passband magnitude response equal to $H(\omega) = \text{sinc}(C\omega)^{-P}$. C is the sinc frequency factor, and P is the sinc power. Similarly, you can design inverse sinc highpass filters with a passband magnitude response equal to $H(\omega) = \text{sinc}(C(1-\omega))^{-P}$. For both the highpass and lowpass filters, the default values of C and P are set to 0.5 and 1, respectively. For more information about the sinc frequency factor and sinc power design options, see the corresponding reference topics.

New Inverse Sinc Highpass Filter Designs

This release adds support for designing highpass inverse sinc filters in MATLAB and Simulink. This capability is available through a new `fdesign.isinchp` filter specification object as well as a new `isinchp` filterbuilder response type.

Filterbuilder and dspfdesign Library Blocks Now Support Different Numerator and Denominator Orders for IIR Filters

As of R2011b, you can now specify different numerator and denominator orders for IIR filters designed using certain filter responses. This capability is available for the

bandpass, bandstop, highpass, and lowpass filter responses in the `filterbuilder` GUI and the corresponding `dspfdesign` library blocks.

New Stopband Shape and Stopband Decay Design Options for Equiripple Highpass Filter Designs

This release adds new stopband shape and stopband decay design options in MATLAB and Simulink. These options are available through the `fdesign.highpass` filter specification object, the `highpass` `filterbuilder` response type, and the Highpass Filter block in the `dspfdesign` library.

FFTW Library Support for Non-Power-of-Two Transform Length

The FFT, IFFT blocks, and the `dsp.IFFT`, `dsp.FFT` System objects include the use of the FFTW library. The blocks and objects now support non-power-of-two transform lengths.

MATLAB Compiler Support for `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter`

R2011b adds MATLAB Compiler support for the `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System objects. With this capability, you can use the MATLAB Compiler to take MATLAB files, which can include System objects, as input and generate standalone applications.

Complex Input Support for `dsp.DigitalDownConverter`

The `dsp.DigitalDownConverter` System object now supports complex inputs.

`getFilters` Method of `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` Now Return Actual Fixed-Point Settings

You can now access the actual fixed-point settings of the filter being used by the `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System objects. To do so, you must first provide a fixed-point input to the object using the `step` method. Then, after the object is locked, call the `getFilters` method to access the actual fixed-point properties of the filter being implemented by the System object. Calling `getFilters` on

an unlocked `dsp.DigitalDownConverter` or `dsp.DigitalUpConverter` System object returns the same results as previous releases.

dsp.SineWave and dsp.BiquadFilter Properties Not Tunable

The following `dsp.SineWave` properties are now nontunable:

- `Frequency`
- `PhaseOffset`

The following `dsp.BiquadFilter` properties are now nontunable:

- `SOSMatrix`
- `ScaleValues`

When objects are locked (i.e., after calling the `step` method), you cannot change any nontunable property values.

Compatibility Considerations

Review any code that changes any `dsp.SineWave` or `dsp.BiquadFilter` property value after calling the `step` method. You should update the code to use property values that do not change.

System Object `DataType` and `CustomDataType` Properties Changes

When you set a System object, fixed-point `<xxx>DataType` property to 'Custom', it activates a dependent `Custom<xxx>DataType` property. If you set that dependent `Custom<xxx>DataType` property before setting its `<xxx>DataType` property, a warning message displays. `<xxx>` differs for each object.

Compatibility Considerations

Previously, setting the dependent `Custom<xxx>DataType` property would automatically change its `<xxx>DataType` property to 'Custom'. If you have code that sets the dependent property first, avoid warnings by updating your code. Set the `<xxx>DataType` property to 'Custom' before setting its `Custom<xxx>DataType` property.

Note If you have a `Custom<xxx>DataType` in your code, but do not explicitly update your code to change `<xxx>DataType` to `'Custom'`, you may see different numerical output.

System Objects Variable-Size Input Dimensions

System objects that process variable-size input now also accept inputs where the number of input dimensions change.

Conversion of Error and Warning Message Identifiers

R2011b changes some error and warning message identifiers in DSP System Toolbox software. For System objects, both error and warning message identifiers have changed. On the Simulink side, the Time Scope block has one warning message with a new identifier.

Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages. You can also use them in code that uses a `try/catch` statement and performs an action based on a specific error identifier.

For example, for System objects, the `MATLAB:system:System:inputSpecsChangedWarning` identifier has changed to `MATLAB:system:inputSpecsChangedWarning`. If your code checks for `MATLAB:system:System:inputSpecsChangedWarning`, you must update it to check for `MATLAB:system:inputSpecsChangedWarning` instead.

For the Time Scope block, the `Simulink:Engine:Simulink:Engine:UnableToUpdateDisplayInRapidAccelerate` identifier has changed to `Simulink:Engine:UINotUpdatedDuringRapidAccelerateSim`. If your code checks for `Simulink:Engine:Simulink:Engine:UnableToUpdateDisplayInRapidAccelerate`, you must update it to check for `Simulink:Engine:UINotUpdatedDuringRapidAccelerateSim` instead.

To determine the identifier for a warning, run the following command just after you see the warning:

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable *MSGID*.

To determine the identifier for an error that appears at the MATLAB prompt, run the following command just after you see the error:

```
exception = MException.last;  
MSGID = exception.identifier;
```

Warning messages indicate a potential issue with your code. While you can turn off a warning, a suggested alternative is to change your code so it runs without warnings.

New and Updated Demos

R2011b adds the following new demos:

- 3-Band Parametric Audio Equalizer Using UDP Packets and Code Generation — Provides a three-band parametric equalizer algorithm based in MATLAB. The demo allows you to dynamically adjust the coefficients of the filters and shows you how to use the MATLAB Coder product to build a standalone executable file that you can run outside of MATLAB.
- Creating New Kinds of System Objects for File Input and Output — Provides an example of creating custom System objects in MATLAB for file input and output.

Additionally, this release updates the IIR Filter Design Given a Prescribed Group Delay demo to use the new `fdesign.arbgrpdelay` object.

Blocks Being Removed in a Future Release

The following blocks will be removed from the DSP System Toolbox product in a future release.

Block Being Removed (library)	Replacement Block
Digital FIR Filter Design (dspddes3)	Discrete FIR Filter
Remez FIR Filter Design (dspddes3)	Discrete FIR Filter
Least Squares FIR Filter Design (dspddes3)	Discrete FIR Filter

Block Being Removed (library)	Replacement Block
Digital FIR Raised Cosine Filter Design (dspddes3)	Discrete FIR Filter
Digital IIR Filter Design (dspddes3)	Discrete Filter
Yule-Walker IIR Filter Design (dspddes3)	Discrete Filter
Integer Delay (dspobslib)	Delay
Dyadic Analysis Filter Bank (dspobslib)	Dyadic Analysis Filter Bank (dspmlti4)
Dyadic Synthesis Filter Bank (dspobslib)	Dyadic Synthesis Filter Bank (dspmlti4)
Wavelet Analysis (dspobslib)	DWT
Wavelet Synthesis (dspobslib)	IDWT
Direct-Form II Transpose Filter (dsparch3)	Digital Filter
Time-Varying Direct-Form II Transpose Filter (dsparch3)	Digital Filter, Discrete FIR Filter, or Allpole Filter
Time-Varying Lattice Filter (dsparch3)	Digital Filter, Discrete FIR Filter, or Allpole Filter

Compatibility Considerations

Beginning in R2011b, Simulink will generate a warning when you load a model that contains one or more of the blocks listed in the preceding table. To ensure that your models continue to work as expected when these blocks are removed from the product in a future release, it is strongly recommended that you replace these unsupported blocks as soon as possible. You can automatically update the blocks in your model by using the `slupdate` function.

R2011a

Version: 8.0

New Features

Bug Fixes

Compatibility Considerations

Product Restructuring

The DSP System Toolbox product replaces the Signal Processing Blockset™ and Filter Design Toolbox™ products in R2011a.

You can access archived documentation for the Signal Processing Blockset and Filter Design Toolbox products on the MathWorks Web site.

Frame-Based Processing

In signal processing applications, you often need to process sequential samples of data at once as a group, rather than one sample at a time. DSP System Toolbox documentation refers to the former as frame-based processing and the latter as sample-based processing. A frame is a collection of samples of data, sequential in time.

Historically, Simulink-family products that can perform frame-based processing propagate frame-based signals throughout a model. The frame status is an attribute of the signals in a model, just as data type and dimensions are attributes of a signal. The Simulink engine propagates the frame attribute of a signal by means of a frame bit, which can either be on or off. When the frame bit is on, Simulink interprets the signal as frame based and displays it as a double line, rather than the single line sample-based signal.

General Product-Wide Changes

Beginning in R2010b, MathWorks started to significantly change the handling of frame-based processing. In the future, frame status will no longer be a signal attribute. Instead, individual blocks will control whether they treat inputs as frames of data or as samples of data. To learn how a particular block handles its input, you can refer to the block reference page.

To make the transition to the new paradigm of frame-based processing, many blocks have received new parameters. You can view an example of how to use these parameters to control sample- and frame-based processing in R2011a and future releases. To open the model, type `ex_inputprocessing` at the MATLAB command line. This model demonstrates how a block can process a signal as sample based or frame based, depending on the setting of that block's **Input processing** parameter.

Notice that when the Discrete FIR Filter and Time Scope blocks are configured to perform frame-based processing, they interpret columns as channels and treat the 2-by-2 input signal as two independent channels. Conversely, when the blocks are configured to

perform sample-based processing, they interpret elements as channels and treat the 2-by-2 input signal as four independent channels. For further information about sample- and frame-based processing, see [Sample- and Frame-Based Concepts](#).

The following sections provide more detailed information about the specific R2011a DSP System Toolbox software changes that are helping to enable the transition to the new way of frame-based processing:

- “Blocks with a New Input Processing Parameter” on page 14-4
- “Changes to the Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT Blocks” on page 14-6
- “Difference Block Changes” on page 14-7
- “Signal To Workspace Block Changes” on page 14-7
- “Spectrum Scope Block Changes” on page 14-8
- “Sample-Based Row Vector Processing Changes” on page 14-8

Compatibility Considerations

During this transition to the new way of handling frame-based processing, both the old way (frame status as an attribute of a signal) and the new way (each block controls whether to treat inputs as samples or as frames) will coexist for a few releases. For now, the frame bit will still flow throughout a model, and you will still see double signal lines in your existing models that perform frame-based processing.

- **Backward Compatibility** — By default, when you load an existing model in R2011a any new parameters related to the frame-based processing change will be set to their backward-compatible option. For example, if any blocks in your existing models received a new **Input processing** parameter this release, that parameter will be set to `Inherited` (this choice will be removed - see release notes) when you load your model in R2011a. This setting enables your existing models to continue working as expected until you upgrade them. Because the inherited option will be removed in a future release, you should upgrade your existing models as soon as possible.
- **slupdate Function** — To upgrade your existing models to the new way of handling frame-based processing, you can use the `slupdate` function. Your model must be compilable in order to run the `slupdate` function. The function detects all blocks in your model that are in need of updating, and asks you whether you would like to upgrade each block. If you select yes, the `slupdate` function updates your blocks accordingly.

- **Timely Update to Avoid Unexpected Results** — It is important to update your existing models as soon as possible because the frame bit will be removed in a future release. At that time, any blocks that have not yet been upgraded to work with the new paradigm of frame-based processing will automatically transition to perform their library default behavior. The library default behavior of the block might not produce the results you expected, thus causing undesired results in your models. Once the frame bit is removed, you will no longer be able to upgrade your models using the `slupdate` function. Therefore, you should upgrade your existing modes using `slupdate` as soon as possible.

For more detailed information about the specific compatibility considerations related to the R2011a frame-based processing changes, see the following Compatibility Considerations sections.

Blocks with a New Input Processing Parameter

Some DSP System Toolbox blocks are able to process both sample- and frame-based signals. After the transition to the new way of handling frame-based processing, signals will no longer carry information about their frame status. Blocks that can perform both sample- and frame-based processing will require a new parameter that allows you to specify the appropriate processing behavior. To prepare for this change, many blocks are receiving a new **Input processing** parameter. You can set this parameter to `Columns as channels (frame based)` or `Elements as channels (sample based)`, depending upon the type of processing you want. The third choice, `Inherited` (this choice will be removed - see release notes), is a temporary selection. This additional option will help you to migrate your existing models from the old paradigm of frame-based processing to the new paradigm.

For a list of blocks that received a new **Input processing** parameter in R2011a, expand the following list.

Blocks with the New Input Processing Parameter

- Arbitrary Response Filter
- Bandpass Filter
- Bandstop Filter
- CIC Compensator
- CIC Filter
- Comb Filter

-
- Differentiator Filter
 - Halfband Filter
 - Highpass Filter
 - Hilbert Filter
 - Inverse Sinc Filter
 - Lowpass Filter
 - Nyquist Filter
 - Octave Filter
 - Parametric Equalizer
 - Peak-Notch Filter
 - Pulse Shaping Filter
 - Unwrap

For a list of blocks that received an **Input processing** parameter in R2010b, see the R2010b Signal Processing Blockset Release Notes.

Compatibility Considerations

When you load an existing model R2011a, any block with the new **Input processing** parameter will show a setting of `Inherited` (this choice will be removed - see release notes). This setting enables your existing models to continue to work as expected until you upgrade them. Although your old models will still work when you open and run them in R2011a, you should upgrade them as soon as possible.

You can upgrade your existing models, using the `slupdate` function. The function detects all blocks that have `Inherited` (this choice will be removed - see release notes) selected for the **Input processing** parameter, and asks you whether you would like to upgrade each block. If you select yes, the function detects the status of the frame bit on the input port of the block. If the frame bit is 1 (frames), the function sets the **Input processing** parameter to `Columns` as channels (frame based). If the bit is 0 (samples), the function sets the parameter to `Elements` as channels (sample based).

In a future release, the frame bit and the `Inherited` (this choice will be removed - see release notes) option will be removed. At that time, the **Input processing** parameter in models that have not been upgraded will automatically be set

to either `Columns` as channels (frame based) or `Elements` as channels (sample based), depending on the library default setting for each block. If the library default setting does not match the parameter setting in your model, your model will produce unexpected results. Additionally, after the frame bit is removed, you will no longer be able to upgrade your models using the `slupdate` function. Therefore, you should upgrade your existing modes using `slupdate` as soon as possible.

Changes to the Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT Blocks

R2011a updates the Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT blocks to the use new way of frame-based processing. In previous releases, the frame status of the input signal determined how these blocks processed the input. In R2011a, the default behavior of these blocks is to always perform frame-based processing.

Unless you specify otherwise, these blocks now treat each column of the input signal as an individual channel, regardless of its frame status. You can now enable the behavior change in these blocks while still allowing for backward compatibility. This release adds a **Treat Mx1 and unoriented sample-based signals as** parameter for this purpose. This parameter will be removed in a future release, at which point the blocks will always perform frame-based processing.

Compatibility Considerations

The **Treat Mx1 and unoriented sample-based signals as** parameter will be removed in a future release. From that point, the Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT blocks will always perform frame-based processing.

You can use the `slupdate` function to upgrade your existing models that contain one of these blocks. The function detects all Overlap-Add FFT Filter, Overlap-Save FFT Filter, and Short-Time FFT blocks in your model. Then, if you allow it to, `slupdate` performs the following actions:

- If the input to the block is an M -by-1 or unoriented sample-based signal, the `slupdate` function:
 - Places a Transpose block in front of the affected block in your model. This block transposes the M -by-1 or unoriented sample-based input into a 1-by- M row vector. By converting the input to a row vector, the block continues to produce the same results as in previous releases (an M_o -by- M_i output).
 - Sets the **Treat Mx1 and unoriented sample-based signals as** parameter to `One channel`. This setting ensures that your model will continue to produce the

same results when the **Treat Mx1 and unoriented sample-based signals as** parameter is removed in a future release.

- If the input to the block is *not* an M -by-1 or unoriented sample-based signal, the `slupdate` function sets the **Treat Mx1 and unoriented sample-based signals as** parameter to `One channel`. This setting does not affect the behavior of your current model. However, the change does ensure that your model will continue to produce the same results when the **Treat Mx1 and unoriented sample-based signals as** parameter is removed in a future release.

Difference Block Changes

R2011a adds a new **Running difference** parameter to the Difference block.

Compatibility Considerations

In a future release, the following option for the **Running difference** parameter will be removed: `Inherit from input` (this choice will be removed – see release notes). From this time forward, you must specify whether or not the block computes a running difference; the block will no longer make that choice based on the status of the frame bit.

You can use the `slupdate` function to upgrade your existing models that contain a Difference block. The function detects whether your models contain any Difference blocks with the **Running difference** parameter set to `Inherit from input` (this choice will be removed – see release notes). If you do, the function detects the status of the frame bit, and sets the **Running difference** parameter accordingly.

Signal To Workspace Block Changes

R2011a updates the Signal To Workspace block. The block now allows you to choose an output format using the **Save format** parameter. You can choose to save your data as an Array, Structure, or `Structure with time`.

Additionally, the old **Frames** parameter has been replaced by a new **Save 2-D signals as** parameter. This parameter allows you to specify whether the block saves 2-D signals as a 2-D array, or as a 3-D array. To provide for backward compatibility, the **Save 2-D signals as** parameter also has an option `Inherit from input` (this choice will be removed – see release notes). When you select this option, the block saves sample-based data as a 3-D array and frame-based data as a 2-D array.

Compatibility Considerations

In a future release, the following option will be removed: `Inherit from input` (this choice will be removed – see release notes). From this time forward, you must specify whether the block saves signals as a 2-D or 3-D array. The block will no longer make that choice based on the status of the frame bit.

You can use the `slupdate` function to upgrade your existing models that contain a Signal To Workspace block. The function detects whether your models contain any Signal To Workspace blocks with the **Save 2-D signals as** parameter set to `Inherit from input` (this choice will be removed – see release notes). If you do, the function detects the status of the frame bit and sets the **Save 2-D signals as** parameter accordingly.

- If the input signal is frame based, the function sets the **Save 2-D signals as** parameter to 2-D array (concatenate along first dimension).
- If the input signal is sample based, the function sets the **Save 2-D signals as** parameter to 3-D array (concatenate along third dimension).

Spectrum Scope Block Changes

R2011a updates the Spectrum Scope block to use the new way of frame-based processing. To enable this change, the block received a new **Treat Mx1 and unoriented sample-based signals as** parameter. This new parameter is available only when you select the **Buffer input** check box. By default, the new parameter is set to `One channel`. In this mode, the block treats M -by-1 and unoriented sample-based input as a single column vector and buffers the input along that column.

Sample-Based Row Vector Processing Changes

In previous releases, some DSP System Toolbox blocks handled sample-based row vector inputs in a special way. Of the blocks that can treat sample-based row vector inputs differently, there are two categories:

- Some blocks have a **Treat sample-based row input as a column** check box which allows you to explicitly specify how the block should treat sample-based row vector inputs. Expand the following section for a full list of these blocks.

Blocks with a Check Box

- Maximum

-
- Mean
 - Median
 - Minimum
 - Normalization
 - RMS
 - Standard Deviation
 - Variance
 - Other blocks automatically treat a sample-based row vector input as a single channel (column vector). Expand the following section for a full list of these blocks.

Blocks That Implicitly Treat Sample-Based Row Vectors as a Single Channel

- Autocorrelation
- Autocorrelation LPC
- Burg AR Estimator
- Burg Method
- Complex Cepstrum
- Convolution
- Correlation
- Covariance AR Estimator
- Covariance Method
- DCT
- FFT
- IDCT
- IFFT
- Interpolation
- Levinson-Durbin
- LPC to LSF/LSP Conversion
- LPC to/from Cepstral Coefficients
- LPC to/from RC
- LPC/RC to Autocorrelation

- LSF/LSP to LPC Conversion
- Modified Covariance AR Estimator
- Modified Covariance Method
- Peak Finder
- Polynomial Stability Test
- Real Cepstrum
- Sort
- Window Function
- Yule-Walker AR Estimator
- Yule-Walker Method

The special treatment of sample-based row vector inputs will be removed in a future release. See the compatibility considerations for more information about how this change will affect your models.

Compatibility Considerations

The blocks listed will continue to work as expected in R2011a. However, in a future release these blocks will produce a warning when you provide them with a sample-based row vector input, and eventually, their behavior will change.

You can prepare your models for the upcoming change by running the `slupdate` function. If the function detects any blocks that have a **Treat sample-based row input as a column** check box, it performs the following actions:

- If the input to the block is a sample-based row vector, and the **Treat sample-based row input as a column** check box is selected, the `slupdate` function places a Transpose block in front of the affected block. The Transpose block transposes the sample-based row vector into a column vector, which is then input into the affected block. Transposing the input signal ensures that your model will produce the same results in future releases.
- If the **Treat sample-based row input as a column** check box is not selected, or if the input to the block is not a sample-based row vector, the `slupdate` function takes no action. Your model will continue to work as expected in future releases.

If the `slupdate` function detects any blocks that automatically treat sample-based row vectors as a column, it performs the following actions:

-
- If the input to the block is a sample-based row vector, the `slupdate` function places a Transpose block in front of the affected block. The Transpose block transposes the sample-based row vector into a column vector, which is then input into the affected block. Transposing the input signal ensures that your model will produce the same results in future releases.
 - If the input to the block is not a sample-based row vector, the `slupdate` function takes no action. Your model will continue to work as expected in future releases.

New Function for Changing the System Object Package Name from `signalblks` to `dsp`

In R2010b, the package name of Signal Processing Blockset™ System objects changed from `signalblks` to `dsp`. In R2011a, a new function is available to help you update your code. You can use the `sysobjupdate` function to recursively search a folder and its subfolders for MATLAB files that contain System object packages, classes, and properties that have been renamed.

Compatibility Considerations

If you have any existing System object code that uses a package name of `signalblks`, you should use the `sysobjupdate` function to update your code. For more information, type `help sysobjupdate` at the MATLAB command line.

New Discrete FIR Filter Block

R2011a adds a new Discrete FIR Filter block to the DSP System Toolbox Filtering/Filter Implementations library. The block is an implementation of the Simulink Discrete FIR Filter block.

New Printing Capability from the Time Scope Block

You can now print the data you see in the Time Scope block. To send the data to your printer, select **File > Print ...** from the scope menu. To print the data to a MATLAB figure, select **File > Print to Figure**.

Improved Display Updates for the Time Scope Block and System Object

R2011a introduces the capability to improve the performance of the Time Scope block and `dsp.TimeScope` System object by reducing the frequency with which the display updates. You can now choose between this new enhanced performance mode and the old behavior by selecting **Reduce Updates to Improve Performance** from the **Simulation** menu of the block, or the **Playback** menu of the System object. By default, both the block and System object operate in the new enhanced performance mode.

New Implementation Options Added to Blocks in the Filter Designs Library

This release provides filter customization options for blocks in the Filtering/Filter Designs library. You can access these options in the **Filter implementation** section of the block dialog box:

- Implement designed filters as Simulink basic elements or as a digital filter.
- Customize filters built using Simulink basic elements using the **Optimizations** parameters.

Blocks in the Filtering/Filter Designs library also support **Input processing** and **Rate options** parameters in R2011a. For more information, see “Blocks with a New Input Processing Parameter” on page 14-4.

Compatibility Considerations

- Frame-based processing and filters with algebraic loops — For filters that contain sample-by-sample feedback, using a lumped-element implementation instead of Simulink basic elements can eliminate algebraic loops. For supported blocks, use the `slupdate` function on older models with designed filters to convert the designed filters into lumped filters. You can enable this feature manually by clearing the **Use basic elements for filter customization** check box.

For filters with algebraic loops that do not have this option, specify sample-based processing by setting the **Input processing** parameter to `Elements as channels (sample based)`.

-
- **Rate Options** parameter — Filters that allow multirate processing, such as FIR decimators and interpolators, perform single-rate processing by default. For more information, see the block reference pages.

New `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System Objects

This release adds new `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System objects. The digital up converter (DUC) and digital down converter (DDC) System objects provide tools to design interpolation/decimation filters and simplify the steps required to implement the up/down conversion process.

Improved Performance of FFT Implementation with FFTW library

The FFT, IFFT blocks include the use of the FFTW library.

Variable-Size Support for System Objects

The following System objects support inputs that change their size at runtime.

- `dsp.ArrayVectorAdder`
- `dsp.ArrayVectorDivider`
- `dsp.ArrayVectorMultiplier`
- `dsp.ArrayVectorSubtractor`
- `dsp.FFT`
- `dsp.IFFT`
- `dsp.Maximum`
- `dsp.Mean`
- `dsp.Minimum`
- `dsp.Normalizer`
- `dsp.RMS`
- `dsp.StandardDeviation`
- `dsp.UDPReceiver`
- `dsp.UDPSender`

- `dsp.Variance`

Compatibility Considerations

For the `dsp.UDPSender` and `dsp.UDPReceiver` System objects only, you should update your code to stop sending or receiving any data length settings. Support for variable-size data makes the data length settings redundant. For example,

```
% Change these lines to remove explicit lengths:
    step(hudps, dataSent, dataLength);
    [dataReceived len] = step(hudpr);
    bytesReceived = bytesReceived + ...
        length(dataReceived) len;

% Code lines with lengths removed:
    step(hudps, dataSent);
    [dataReceived] = step(hudpr);
    bytesReceived = bytesReceived + ...
        length(dataReceived);
```

System Objects FullPrecisionOverride Property Added

A `FullPrecisionOverride` property has been added to the System objects listed below. This property is a convenient way to control whether the object uses full precision to process fixed-point input.

When you set this property to `true`, which is the default, it eliminates the need to set many fixed-point properties individually. It also hides the display of these properties (such as `RoundingMode`, `OverflowAction`, etc.) because they are no longer applicable individually.

To set individual fixed-point properties, you must first set `FullPrecisionOverride` to `false`.

Note The `CoefficientDataType` property is not controlled by `FullPrecisionOverride`

The following System objects are affected:

-
- `dsp.ArrayVectorAdder`
 - `dsp.ArrayVectorSubtractor`
 - `dsp.Autocorrelator`
 - `dsp.Convolver`
 - `dsp.Crosscorrelator`
 - `dsp.FIRDecimator`
 - `dsp.FIRInterpolator`
 - `dsp.FIRRateConverter`
 - `dsp.SubbandAnalysisFilter`
 - `dsp.SubbandSynthesisFilter`
 - `dsp.Window`

Compatibility Considerations

All of these System objects have their new `FullPrecisionOverride` property set to the default, `true`. If you had set any fixed-point properties to non-default values for these objects, those values are ignored. As a result, you may see different numerical answers from those answers in a previous release. To use your nondefault fixed-point settings, you must first change `FullPrecisionOverride` to `false`.

'Internal rule' System Object Property Value Changed to 'Full precision'

To clarify the value of many `DataType` properties, the 'Internal rule' option has been changed to 'Full precision'.

Compatibility Considerations

The objects allow you to enter either 'Internal rule' or 'Full precision'. If you enter 'Internal rule', that option is stored as 'Full precision'.

MATLAB Compiler Support for System Objects

The DSP System Toolbox supports the MATLAB Compiler for most System objects. With this capability, you can use the MATLAB Compiler to take MATLAB files, which can include System objects, as input and generate standalone applications.

The following System objects are not supported by the MATLAB Compiler software:

- `dsp.CICDecimator`
- `dsp.CICInterpolator`
- `dsp.DigitalDownConverter`
- `dsp.DigitalUpConverter`
- `dsp.TimeScope`

Viewing System Objects in the MATLAB Variable Editor

The MATLAB Variable Editor now displays System objects properties in the same order as they display at the command line. Note that the Variable Editor provides a read-only view for System objects.

System Object Input and Property Warnings Changed to Errors

When a System object is locked (e.g., after the `step` method has been called), the following situations now produce an error. This change prevents the loss of state information.

- Changing the input data type
- Changing the number of input dimensions
- Changing the input complexity from real to complex
- Changing the data type, dimension, or complexity of tunable property
- Changing the value of a nontunable property

Compatibility Considerations

Previously, the object issued a warning for these situations. The object then unlocked, reset its state information, relocked, and continued processing. To update existing code so that it does not produce an error, use the `release` method before changing any of the items listed above.

New and Updated Demos

R2011a adds the following new demos:

-
- **Digital Up and Down Conversion for Family Radio Service** — Shows you how to use the new `dsp.DigitalDownConverter` and `dsp.DigitalUpConverter` System objects to design a Family Radio Service (FRS) transmitter and receiver.
 - **Design and Analysis of a Digital Down Converter** — Shows you how to use the `dsp.DigitalDownConverter` System object to simplify the steps required to emulate the TI Graychip 4016 digital down converter.
 - **Using System Objects with MATLAB Compiler** — Shows you how to use MATLAB Compiler to create a standalone application from MATLAB System objects.

Additionally, the Simulink-based demo, GSM Digital Down Converter, has been enhanced to use the Fixed-Point Toolbox™ `cordicrotate` function. The demo now allows you to compare an NCO-based mixer to a CORDIC-based mixer.

Documentation Examples Renamed

In previous releases, the example models used throughout the Signal Processing Blockset™ documentation were named with a prefix of `doc_`. In R2011a, this prefix has changed to `ex_`. For example, in R2010b, you could launch an example model using the Time Scope block by typing `doc_timescope_tut` at the MATLAB command line. To launch the same model in R2011a, you must type `ex_timescope_tut` at the command line.

Compatibility Considerations

You can no longer launch DSP System Toolbox documentation example models using the `doc_` name. To open these models in R2011a, you must replace the `doc_` prefix in the model name with `ex_`.

Downsample Block No Longer Has Frame-Based Processing Latency for a Frame Size of One

As of R2011a, the Downsample block no longer exhibits frame-based processing latency when the input frame size is one.

Compatibility Considerations

Existing models that use the Downsample block in frame-based processing mode may produce different results in R2011a. Specifically, the Downsample block no longer has

one-frame of latency when the input frame size is one. If your model uses a `Downsample` block in frame-based processing mode and the input frame size is one, you will see different results when you run your model in R2011a. If you need to restore the one-frame latency, you can use a `Delay` block to delay the output of the `Downsample` block by one frame.

SignalReader System Object Accepts Column Input Only

The `SignalReader` System object now accepts column inputs only.

Compatibility Considerations

Update any code with row input to the `SignalReader` object to convert the input to column form before passing it to the object. (Note that this change occurred in R2010b.)

FrameBasedProcessing Property Removed from the dsp.DelayLine and dsp.Normalizer System Objects

In R2010b, the `FrameBasedProcessing` property was removed from the `dsp.DelayLine` and `dsp.Normalizer` System objects. Both objects now treat each column of the input as a separate channel (frame-based processing).

Compatibility Considerations

As of R2010b, MATLAB issues a warning when you set the `FrameBasedProcessing` property of the `dsp.DelayLine` or `dsp.Normalizer` System objects.

R2010a MAT Files with System Objects Load Incorrectly

If you saved a System object to a MAT file in R2010a and load that file in R2011a, MATLAB may display a warning that the constructor must preserve the class of the returned object. This occurs because an aspect of the class definition changed for that object in R2011a. The object's saved property settings may not restore correctly.

Compatibility Considerations

MAT files containing a System object saved in R2010a may not load correctly in R2011a. You should recreate the object with the desired property values and save the MAT file.

