Release Notes for Communications System Toolbox™

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Release Notes for Communications System Toolbox™

Contents

R2012b

Version: 5.3
New Features: Yes
Bug Fixes: No

Support for C code generation for all System objects in Communications Systems Toolbox

Effective this release, the following System objects provide C code generation:

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

All CPU-based System objects in the Communications System Toolbox™ product generate C code. The GPU-based System objects do not generate C code.

Support for HDL code generation for Reed-Solomon encoder, decoder, and CRC detector blocks

Effective this release, the following blocks provide HDL code generation:

- General CRC Syndrome Detector HDL Optimized
- Integer-Input RS Encoder HDL Optimized
- Integer-Output RS Decoder HDL Optimized

To generate HDL code, you must have an HDL Coder™ license.

Support for HDL code generation for Rectangular QAM and PSK Demodulator System objects

Effective this release, the following System objects provide HDL code generation:

- `comm.BPSKDemodulator`
- `comm.QPSKDemodulator`
- `comm.PSKDemodulator`
- `comm.RectangularQAMDemodulator`

To generate HDL code, you must have an HDL Coder license.

LTE Zadoff-Chu sequence generator function

Communications System Toolbox includes a Zadoff-Chu sequence generator function. This function is useful when modeling 3GPP LTE physical layer characteristics, downlink primary synchronization signals, or the uplink reference signals and random access preamble sequences. For more information, see the `lteZadoffChuSeq` Help page.

LTE downlink shared channel example

This example shows the Downlink Shared Channel (eNodeB to UE) processing of the Long Term Evolution (LTE) physical layer (PHY) specifications developed by the Third Generation Partnership Project (3GPP). LTE-Advanced is one of the candidates for fourth generation (4G) communications systems, approved by the International Telecommunication Union (ITU), with expected downlink peak data rates in excess of 1Gbps (for Release 10 and beyond). Using the Release 10 specifications, this example highlights the multi-antenna transmission scheme that enables such high data rates.

Phase Noise block and System object, specifying phase noise spectrum with a vector of frequencies

The Phase Noise block and System object™ now have more flexibility for specifying spectral noise characteristics. You can specify a vector of phase noise levels, at more than one frequency value. Previously, the software allowed the specification of a single-phase noise level point. The new implementation enables more realistic noise modeling in your communications models, and allows you to visualize the phase noise spectrum that the block or System object generates.

IEEE 802.11 beacon with captured data example

This example shows reception of beacon frames in an 802.11 wireless local area network (WLAN). You can select one of several captured signals and view the data the beacon frame carries.

P25 spectrum sensing example

This example shows how to use cyclostationary feature detection to distinguish signals with different modulation schemes, including P25 signals. It defines four cases of signals: noise only, C4FM, CQPSK, and one arbitrary type. The example applies the detection algorithm to signals with different SNR values and determines when the signals can be classified as one of the four types.

MATLAB-based QPSK transceiver example

The QPSK Transmitter and Receiver example now includes a MATLAB® implementation that uses System objects. This example models a digital communications system to simulate the QPSK transmitter - receiver chain. In particular, this example illustrates a method for tackling real-world wireless communication issues, such as: carrier frequency/phase offset, timing recovery, and frame synchronization.

Design Iteration Workflow

This example illustrates a design workflow and the typical iterations involved in designing a wireless communications system with the Communications System Toolbox. Because Communications System Toolbox supports both MATLAB and Simulink®, this examples showcases separate design iterations using MATLAB functions or Simulink models.

The workflow starts with a simple QPSK modulator system that transmits a signal through an AWGN channel and calculates the bit error rate. To make the system more realistic and improve system performance, the example gradually introduces Viterbi decoding, turbo coding, multipath fading channels, OFDM-based transmission and equalization, and multiple-antenna techniques.

`Constellation` method for modulator and demodulator System objects

Effective this release, modulator and demodulator System object have a `constellation` method. This method calculates or plots the ideal signal constellation, depending on object settings. The following System objects have the `constellation` method:

- `comm.PSKModulator`
- `comm.PSKDemodulator`
- `comm.RectangularQAMModulator`
- `comm.RectangularQAMDemodulator`
- `comm.PAMModulator`
- `comm.PAMDemodulator`
- `comm.QPSKModulator`
- `comm.QPSKDemodulator`
- `comm.BPSKModulator`
- `comm.BPSKDemodulator`
- `comm.OQPSKModulator`
- `comm.OQPSKDemodulator`
- `comm.gpu.PSKModulator`
- `comm.gpu.PSKDemodulator`

Specify initial states of Gold Sequence Generator and PN Sequence Generator System objects

You can specify the initial states for the PN Sequence Generator and Gold Sequence Generator System objects as inputs to the `step` method. You can use these System objects as scrambling sequence generators. For packet-based systems, including WiMAX and LTE, the initial conditions are a function of time. Therefore, for simulation purposes, you must specify the initial states as an input.

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 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

Compatibility Considerations: Yes

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.

Communications System Toolbox Functionality Being Changed or Removed

Compatibility Considerations: Yes

The following function, which was previously announced for removal and warned at run time, has been removed from the product.

- seqgen.pn

The following functions will be removed in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`commmeasure.ACPR`	Warns	`comm.ACPR`	Replace all instances of `commmeasure.ACPR` with `comm.ACPR`.
`commmeasure.EVM`	Warns	`comm.EVM`	Replace all instances of `commmeasure.EVM` with `comm.EVM`.
`commmeasure.MER`	Warns	`comm.MER`	Replace all instances of `commmeasure.MER` with `comm.MER`.
`fec.bchdec`	Warns	`comm.BCHDecoder`	Replace all instances of `fec.bchdec` with `comm.BCHDecoder`.
`fec.bchenc`	Warns	`comm.BCHEncoder`	Replace all instances of `fec.bchenc` with `comm.BCHEncoder`.
`fec.ldpcdec`	Warns	`comm.LDPCDecoder`	Replace all instances of `fec.ldpcdec` with `comm.LDPCDecoder`.
`fec.ldpcenc`	Warns	`comm.LDPCEncoder`	Replace all instances of `fec.ldpcenc` with `comm.LDPCEncoder`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`fec.rsdec`	Warns	`comm.RSDecoder`	Replace all instances of `fec.rsdec` with `comm.RSDecoder`.
`fec.rsenc`	Warns	`comm.RSEncoder`	Replace all instances of `fec.rsenc` with `comm.RSEncoder`.

Update Legacy Code to use System objects

For help updating your legacy code so that it uses the new System objects, refer to the following sections.

Map commmeasure.ACPR Properties and Methods to comm.ACPR

commmseaure.ACPR property	comm.ACPR property	Note
`Fs`	`SampleRate`	
`MainChannelMeasBW`	`MainMeasurementBandwidth`	
`AdjacentChannelMeasBW`	`AdjacentMeasurementBandwidth`	
`MeasurementFilter`	`MeasurementFilterSource`	
`SpectralEstimatorOption`	`SpectralEstimation`	
`WindowOption`	`Window`	
`SidelobeAtten`	`SidelobeAttenuation`	
`FrequencyResolutionOption`	`FrequencyResolution`	
`FFTLength`	`CustomFFTLength`	

commmseaure.ACPR property	**comm.ACPR property**	**Note**
	`MainChannelPowerOutputPort` (new property)	When you set `MainChannelPowerOutputPort` to `true`, the main channel power measurement becomes an output. **Note** Previously, for the `commmeasure.ACPR` object, this was the second output argument.
	`AdjacentChannelPowerOutputPort` (new property)	When you set `AdjacentChannelPowerOutputPort` to `true`, the adjacent channel power measurement becomes an output. **Note** Previously, for the `commmeasure.ACPR` object, this was the third output argument.
`Type`	N/A	This read-only property was removed.
`FrameCount`	N/A	This read-only property was removed.

commmseaure.ACPR method	comm.ACPR method
run	step
reset	reset
copy	clone
disp	N/A

Note commmeasure.ACPR and comm.ACPR have a different API. Refer to the following syntax examples when updating your legacy code:

commmeasure.ACPR	comm.ACPR	Note
commmeasure.ACPR	comm.ACPR	The default settings of the following are different: `NormalizedFrequency' `MainMeasurementBandwidth' `AdjacentChannelOffset' `AdjacentMeasurementBandwidth' `MeasurementFilterSource'

h = commmeasure.ACPR(
'PowerUnits','linear',
'SpectralEstimatorOption','UserDefined',
'SegmentLength',100);
[act_ACPR, actMainPow, actAdjPow] = run(h,yPulse);
fd = h.MeasurementFilter;
	h = comm.ACPR(
	'PowerUnits','Watts',
	'SpectralEstimation', 'Specify window parameters',
	'SegmentLength',100,
	'MainChannelPowerOutputPort', true,
	'AdjacentChannelPowerOutputPort', true,...
	'MeasurementFilterSource', 'property');
	[act_ACPR, actMainPow, actAdjPow] = step(h,yPulse);
	fdnumerator = h.MeasurementFilter;
		MeasurementFilter changes from a structure to a variable.

Map commmeasure.EVM Properties and Methods to comm.EVM

commmseaure.EVM properties	comm.EVM properties	Note
NormalizationOption	Normalization	
AveragePower	AverageConstellationPower	
PeakPower	PeakConstellationPower	
RSMEVM	N/A	RSMEVM is an output.
MaximumEVM	MaximumEVMOutputPort	When you set MaximumEVMOutputPort to true, MaximumEVM becomes an output.
Percentile	XPercentileValue	XPercentileValue appears when you set the XPercentileEVMOutputPort to true.
PercentileEVM	XPercentileEVMOutputPort	When you set XPercentileEVMOutputPort to true, PercentileEVM becomes an output.
NumberOfSymbols	SymbolCountOutputPort	When you set SymbolCountOutputPort to true, NumberOfSymbols becomes an output.
Type	N/A	This read-only property was removed.

commmseaure.EVM methods	comm.EVM methods
update (no outputs)	step (multiple outputs)
reset	reset
copy	clone

Note `commmeasure.evm` and `comm.evm` have a different API. Refer to the following syntax examples when updating your legacy code:

commmeasure.EVM	comm.EVM
`hEVM = commmeasure.EVM('Percentile', 90);`	`hEVM = comm.EVM('XPercentileEVMOutputPort',`

`update(hEVM, rcv, xmv)`
`rmsevm = hEVM.RMSEVM`
	`rmsevm = step(hEVM, rcv, xmv)`

`update(hEVM, rcv, xmv)`
`rmsevm = hEVM.RMSEVM`
`maxevm = hEVM.MaximumEVM`
`pevm =hEVM.PercentileEVM`
`numsym = hEVM. NumberOfSymbols`
	`[rmsevm,maxevm,pevm,numsym] = step(hEVM, rcv`

Map commmeasure.MER Properties and Methods to comm.MER

commmseaure.MER properties	comm.MER properties	Note
`MERdb`	N/A	MERdb is an output.
`MinimumMER`	MinimumMEROutputPort	When you set `MinimumMEROutputPort` to `true`, `MimimumMER` becomes an output.
`Percentile`	`XPercentileValue`	`XPercentileValue` appears when you set the `XPercentileMEROutputPort` to `true`.

commmseaure.MER properties	comm.MER properties	Note
`PercentileMER`	`XPercentileMEROutputPort`	When you set `XPercentileMEROutputPort` to `true`, `PercentileMER` becomes an output.
`NumberOfSymbols`	`SymbolCountOutputPort`	When you set `SymbolCountOutputPort` to `true`, `NumberOfSymbols` becomes an output.
`Type`	N/A	This read-only property was removed.

commmseaure.MER methods	comm.MER methods
`update` (no outputs)	`step` (multiple outputs)
`reset`	`reset`
`copy`	`clone`

Note `commmeasure.MER` and `comm.MER` have a different API. Refer to the following syntax examples when updating your legacy code:

commmseaure.MER	comm.MER
`hMER = commmeasure.MER('Percentile', 90);`	`hMER = comm.MER('XPercentileMEROutputPort', t`
`update(hMER, rcv, xmv)merdb = hMER.MERdB`	`merdb = step(hMER, rcv, xmv)`
`update(hMER, rcv, xmv)` `merdb = hEVM.MERdB`	`[merdb,minimummer,pmer,numsym] = step(hmer, r`

commmseaure.MER	comm.MER

`minimummer = hEVM.MinimumMER`
`pmer = hEVM.PercentileMER`
`numsym = hEVM. NumberOfSymbols`	

Map fec.bchenc Properties to comm.BCHEncoder

fec.bchenc property	comm.BCHEncoder property	Note
`N`	`CodewordLength`	
`K`	`MessageLength`	
`T`	The `ErrorCorrectionCapability` element of the `Info`	
`ShortenedLength`	N/A	This information is included in the `CodewordLength` and `MessageLength` properties.
`ParityPosition`	N/A	Always `'end'`.
`PuncturePattern`	`PuncturePattern`	This property appears when you set
`GenPoly`	`GeneratorPolynomial`	This property appears when you set
`Type`	N/A	This read-only property was removed.

Note `fec.bchenc` and `comm.BCHEncoder` have a different API. Refer to the following syntax examples when updating your legacy code:

fec.bchenc	comm.BCHEncoder	Note
`h=fec.bchenc`	`h = comm.BCHEncoder('CodewordLength',7,'MessageLengt`	Use this syntax to create the default configuration of `fec.bchenc`.

`enc = fec.bchenc(7,4);`
`msg = [0 1 1 0]';`
`code = encode(enc,msg);`
	`h = comm.BCHEncoder('CodewordLength',7,'MessageLengt`
	`msg = [0 1 1 0]';`
	`code = step(h,msg)`
		• `GeneratorPolynomial` must be a column vector and `PuncturePattern` must be a row vector.
		• The `step` method replaces use of the `encode` function.

`encShort = fec.bchenc(7,4);`
`encShort.ShortenedLength = 1;`
`msgShort = [0 1 1]';`
`codeShort = encode(encShort,msgShort);`
	`h=comm.BCHEncoder(6,3);`
	`msg = [0 1 1]';`
	`code = step(h,msg)`
		The shortened length information is included in the `CodewordLength` and `MessageLength` properties.

Map fec.bcdec Properties to comm.BCHDecoder

fec.bchdeproperty	comm.BCHDecoder property	Note
`N`	`CodewordLength`	
`K`	`MessageLength`	
`T`	The `ErrorCorrectionCapability` element of the `Info` method	This information is included in the `CodewordLength` and `MessageLength` properties.
`ShortenedLength`	N/A	
`ParityPosition`	N/A	

fec.bchdeproperty	comm.BCHDecoder property	Note
`PuncturePattern`	`PuncturePattern`	This property appears when you set
`GenPoly`	`GeneratorPolynomial`	This property appears when you set
`Type`	N/A	This read-only property was removed.

Note `fec.bchdec` and `comm.BCHDecoder` have a different API. Refer to the following syntax examples when updating your legacy code:

fec.bchdec	fec.BCHDecoder	Note
`h=fec.bchdec`	`h = comm.BCHDecoder('CodewordLength',7,'MessageLengt`	Use this syntax to create the default configuration of `fec.bchdec`.

`dec = fec.bchdec(7,4);`
`code = [0 1 1 0 0 0 1].';`
`msg = decode(dec,code);`
	`h = comm.BCHDecoder('CodewordLength',7,'MessageLengt`
	`code = [0 1 1 0 0 0 1].';`
	`msg = step(h,code)`
		• `GeneratorPolynomial` must be a column vector and `PuncturePattern` must be a row vector.
		• The `step` method replaces use of the `decode` function.

`decShort = fec.bchdec(7,4)`
`decShort.ShortenedLength = 1;`
`code = [0 1 1 1 0 1].';`
`msg  = decode(decShort,code);`
	`h = comm.BCHDecoder('CodewordLength',6,'MessageLengt`
	`code = [0 1 1 1 0 1]';`
	`msg = step(h,code)`
		The shortened length information is included in the `CodewordLength` and `MessageLength` properties.

Map fec.ldpcenc Properties to comm.LDPCEncoder

fec.ldpcenc property	comm.LDPCEncoder property	Note
`ParityCheckMatrix`	`ParityCheckMatrix`	
`BlockLength`	N/A	This read-only property was removed.
`NumInfoBits`	N/A	This read-only property was removed.
`NumParityBits`	N/A	This read-only property was removed.
`EncodingAlgorithm`	N/A	This read-only property was removed.

Note The `comm.LDPCEncoder` System object does contain all the read-only properties of the old object. However, you can obtain the information from the `ParityCheckMatrix`.

`fec.ldpcenc` and `comm.LDPCEncoder` have a different API. Refer to the following syntax example when updating your legacy code:

fec.ldpcenc	comm.LDPCEEncoder	Note

`h1 = fec.ldpcenc;`
`xin = ones(32400,1);`
`yout1 = encode(h1,xin.')`
	`h = comm.LDPCEncoder;`
	`xin = ones(32400,1);`
	`yout = step(h, xin)`
		• The `fec.ldpcenc` object accepted a row vector input. The `comm.LDPCEncoder` System object accepts a column vector input.
		• The `step` method replaces use of the `encode` function

Map fec.ldpcdec Properties to comm.LDPCDecoder

fec.ldpcdec property	comm.LDPCDecoder property	Note
`ParityCheckMatrix`	`ParityCheckMatrix`	
`DecisionType`	`DecisionMethod`	
`OutputFormat`	`OutputValue`	
`DoParityChecks`	`IterationTerminationCondition`	Select `Parity check satisfied`
`NumIterations`	`MaximumIterationCount`	
`ActualNumIterations`	`NumIterationsOutputPort`	
`FinalParityChecks`	`FinalParityChecksOutputPort`	
`BlockLength`	N/A	This read-only property was removed.
`NumInfoBits`	N/A	This read-only property was removed.
`NumParityBits`	N/A	This read-only property was removed.

Note The `comm.LDPCDecoder` System object does not contain all the read-only properties of the old object. The `ActualNumIterations` and `FinalParityChecks` properties become outputs.

`fec.ldpcdec` and `comm.LDPCDecoder` have a different API. Refer to the following syntax example when updating your legacy code.

fec.ldpcdec	comm.LDPCDecoder	Note

`h1 = fec.ldpcdec;`
`yin = ones(64800,1);`
`yout1 = decode(h1,yin.');`
	`h = comm.LDPCDecoder`
	`yin = ones(64800,1);`
	`yout = step(h,yin)`
		• The `fec.ldpcdec` object accepted a row vector input. The `comm.LDPCDecoder` System object accepts a column vector input.
		• The `step` method replaces use of the `decode` function

Map fec.rsenc Properties to comm.RSEncoder

fec.rsenc	comm.RSEncoder	Note
`N`	`CodewordLength`	
`K`	`MessageLength`	
`T`	The `ErrorCorrectionCapability` element of the `Info`	
`ShortenedLength`	N/A	This information is included in the `CodewordLength` and `MessageLength` properties.
`ParityPosition`	N/A	Always `'end'`.
`GenPoly`	`GeneratorPolynomial`	This property appears when you set `ource` `to Property`.
`Type`	N/A	This read-only property was removed.

Note `fec.rsenc` and `comm.RSEncoder` have a different API. Refer to the following syntax examples when updating your legacy code:

---

fec.rsenc	comm.RSEncoder	Note

`h=fec.rsenc`	`h = comm.RSEncoder('CodewordLength',7,'MessageLength`
	
	`h = comm.BCHEncoder('CodewordLength',7,'MessageLengt`
		Use this syntax to create the default configuration of `fec.rsenc`.

`enc = fec.rsenc(7,3);`
`msg = [0 1 0]';`
`code = encode(enc,msg);`
	`h = comm.RSEncoder('CodewordLength',7,'MessageLength`
	`msg = [0 1 0]';`
	`code = step(h,msg)`
		• `GeneratorPolynomial` must be a column vector and `PuncturePattern`must be a row vector.
		• The `step` method replaces use of the `encode` function.

`encShort = fec.rsenc(7,3);`
`encShort.ShortenedLength = 1;`
`msgShort = [0 1]';`
`codeShort = encode(encShort,msgShort);`
	`h = comm.RSEncoder('CodewordLength',6,'MessageLength`
	`msg = [0 1]';`
	`code = step(h,msg)`
		• The shortened length information is included in the `CodewordLength` and `MessageLength` properties.
		• The `step` method replaces use of the `encode` function.

Map fec.rsdec Properties to comm.RSDecoder

fec.rsdec	comm.RSDecoder	Note
`N`	`CodewordLength`	
`K`	`MessageLength`	
`T`	The `ErrorCorrectionCapability` element of the `Info`	
`ShortenedLength`	N/A	This information is included in the `CodewordLength` and `MessageLength` properties.

fec.rsdec	**comm.RSDecoder**	**Note**
`ParityPosition`	N/A	Always `'end'`.
`PuncturePattern`	`PuncturePattern`	This property appears when you set
`GenPoly`	`GeneratorPolynomial`	This property appears when you set
`Type`	N/A	This read-only property was removed.

Note `fec.rsdec` and `comm.RSDecoder` have a different API. Refer to the following syntax examples when updating your legacy code:

fec.rsdec	**comm.RSDecoder**	**Note**
`h=fec.rsdec`	`h = comm.RSDecoder('CodewordLength',7,'MessageLength`	Use this syntax to create the default configuration of `fec.rsdec`.

`dec = fec.rsdec(7,3);`
`code = [0 1 1 0 0 0 1].';`
`msg = decode(dec,code);`
	`h = comm.RSDecoder('CodewordLength',7,'MessageLength`
	`code = [0 1 1 0 0 0 1].';`
	`msg = step(h,code)`
		• `GeneratorPolynomial` must be a column vector and `PuncturePattern`must be a row vector.
		• The `step` method replaces use of the `encode` function.

`decShort = fec.rsdec(7,3);`
`decShort.ShortenedLength = 1;`
`code = [0 1 1 1 0 1].';`
`msg  = decode(decShort,code);`
	`h = comm.RSDecoder('CodewordLength',6,'MessageLength`
	`code = [0 1 1 1 0 1]';`
	`msg = step(h,code)`
		• The shortened length information is included in the `CodewordLength` and `MessageLength` properties.
		• The `step` method replaces use of the `encode` function.

Frame-Based Processing

Compatibility Considerations: Yes

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. For more information, see “Frame-Based Processing” on page 84.

R2012a

Version: 5.2
New Features: Yes
Bug Fixes: No

MIMO Multipath Fading Channel System Objects

The Communications System Toolbox product now includes a Multiple Input Multiple Output (MIMO) Multipath Fading Channel System object, comm.MIMOChannel. Multipath MIMO fading channels allow for design of communication systems with multiple antenna elements at the transmitter and receiver. For more information, see the `comm.MIMOChannel` Help page.

The product also includes an LTE MIMO Multipath Fading Channel System object, comm.LTEMIMOChannel. This object allows for design of communication systems with multiple antenna elements at the transmitter and receiver using the 3GPP Long Term Evolution (LTE) standard. For more information, see the `comm.LTEMIMOChannel` Help page.

Multi-H Support for CPM Modulator and Demodulator Simulink Blocks and MATLAB System Objects

The CPM Modulator Baseband and CPM Demodulator Baseband blocks and System objects now support Multi-H CPM modulation. These enhancements allow you to perform research and development work for communication systems designed with the ARTM, JTRS, or MIL-STD-188–181C communications standards. For more information, see:

- `comm.CPMModulator`
- `comm.CPMDemodulator`
- CPM Modulator Baseband
- CPM Demodulator Baseband

GPU System Objects

This release adds new GPU System objects, which use a graphics processing unit (GPU) to procure simulation results more quickly than a CPU. These new objects include:

- `comm.gpu.ConvolutionalInterleaver`
- `comm.gpu.ConvolutionalDeinterleaver`
- `comm.gpu.ConvolutionalEncoder`
- `comm.gpu.PSKDemodulator`
- `comm.gpu.TurboDecoder`

MATLAB Compiler Support for GPU System Objects

In Release 2012a, you can use the MATLAB Compiler™ product with GPU System objects. With this capability, MATLAB Compiler software can generate standalone applications from MATLAB files, including files that contain GPU System objects.

Code Generation Support

The following System objects now support C code generation:

- `comm.BCHEncoder`
- `comm.RSEncoder`

The following function now supports C code generation:

- `bchgenpoly`

HDL Code Generation from MATLAB code

The following System objects now support HDL code generation:

- `comm.ViterbiDecoder`
- `comm.PSKModulator`
- `comm.BPSKModulator`
- `comm.QPSKModulator`
- `comm.rectangularQAMmodulator`
- `comm.ConvolutionalInterleaver`
- `comm.ConvolutionalDeinterleaver`

See also HDL Code Generation from MATLAB.

HDL Support For HDL CRC Generator Block

Release R2012a provides HDL code generation support for the new HDL CRC Generator block.

Enhancements for System Objects Defined by Users

Compatibility Considerations: Yes

This release contains enhancements for System objects defined by users.

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 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 runtime.

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 a System object's 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

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.

New and Enhanced Demos

The following demos are new or enhanced for this release:

- IEEE® 802.11 WLAN - Beacon Frame simulates packetized, non-streaming transmission and reception of beacon frames in an 802.11-based wireless local area network (WLAN).
- IEEE® 802.16-2009 WirelessMAN-OFDMA PHY Downlink PUSC simulates a downlink partial usage of subchannels (PUSC) Physical Layer communication from base station (BS) to two mobile stations. This demo uses variable-size signals to model dynamic channel allocation between the two users.
- QPSK Transmitter and Receiver implements a QPSK transmitter and receiver, including carrier and timing recovery.
- Digital Video Broadcasting - Cable (DVB-C) models part of the ETSI (European Telecommunications Standards Institute) EN 300 429 standard for cable system transmission of digital television signals.
- Downlink Transport Channel (DL-SCH) Processing models part of the transport channel processing for the Downlink Shared Channel (eNodeB to UE) of the Long Term Evolution (LTE) specifications developed by the Third Generation Partnership Project (3GPP) .
- Using GPUs To Accelerate Turbo Coding Bit Error Rate Simulations shows how you can use GPUs to dramatically accelerate bit error rate simulations.
- End to End System Simulation Acceleration Using GPUs compares four techniques that can be used to accelerate bit error rate (BER) simulations.

Functionality Being Changed or Removed

Compatibility Considerations: Yes

The following functions will be removed in a future release.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
rsdecof	Warns	comm.RSDecoder	Replace all instances of rsdecof with comm.RSDecoder.
rsencof	Warns	comm.RSEncoder	Replace all instances of rsencof with comm.RSEncoder.

The following functions, which were previously announced for removal in a future release, now warn at run time. You should not use these functions.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
rcosflt	Warns	fdesign.pulseshaping	• Use `fdesign.interpolator` and `fdesign.decimator` to design multirate filters. • Use `fdesign.pulseshaping` to design a single-rate raised cosine filter. Does not support IIR.
rcosiir	Warns	N/A	Do not use.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
rcosine	Warns	fdesignpulseshaping	• Use `fdesign.interpolator` and `fdesign.decimator` to design multirate filters. • Use `fdesign.pulseshaping` to design a single-rate raised cosine filter. Does not support IIR.
bchdec	Warns	comm.BCHDecoder	
bchenc	Warns	comm.BCHEncoder	
rsdec	Warns	comm.RSDecoder	
rsenc	Warns	comm.RSEncoder	
randint	Warns	randi	Use `randi` to generate matrix of uniformly distributed random integers

Several functions, which were previously announced for removal in a future release and warned at run time, have been removed from the Communications System Toolbox product. To see the full list of these removed functions, expand the following section.

Removed Functions

- ademod
- ademodce
- amod
- amodce

- apkconst
- bchdeco
- bchenco
- bchpoly
- constlay
- convdeco
- convenco
- ddemod
- ddemodce
- demodmap
- dmod
- dmodce
- eyescat
- flxor
- gen2abcd
- gfplus
- htruthtb
- imp2sys
- lineprob
- modmap
- oct2gen
- qaskdeco
- qaskenco
- randbit
- rscore
- rsdeco
- rsdecode

- rsenco
- rsencode
- rspoly
- sim2gen
- sim2gen2
- sim2logi
- sim2tran
- simpassbandex
- simsum
- simsum2
- viterbi
- vitshort

The following function, which was previously announced for removal in a future release, will remain in the Communications System Toolbox product.

- `rcosfir`

Frame-Based Processing

Compatibility Considerations: Yes

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. For more information, see "Frame-Based Processing" on page 84.

Inherited Option of the Input Processing Parameter Now Warns

Some Communications 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 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

To see a list of Communications System Toolbox blocks that contain the **Input processing** parameter, expand the following section.

Blocks with Input Processing Parameter

- AWGN Channel (with only two options)
- Derepeat
- Gaussian Filter
- Ideal Rectangular Pulse Filter
- Raised Cosine Receive Filter
- Raised Cosine Transmit Filter
- Windowed Integrator

Compatibility Considerations

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.

Inherited Option of the Rate Options Parameter Now Warns

Some Communications System Toolbox blocks support single-rate or multirate processing. After the transition to the new paradigm for handling frame-based processing, signals will no longer carry information about their frame status. Blocks that can perform both single-rate and multirate processing have a new parameter that allows you to specify the appropriate processing behavior. To prepare for this change, many blocks received a new **Rate options** parameter in previous releases. You can set this parameter to `Enforce single-rate processing` or `Allow multirate processing`. The third choice, `Inherit from input (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 met for any block in your model:

- The **Rate options** parameter set to `Inherit from input (this choice will be removed - see release notes)`
- The input signal is sample-based
- The input signal is a scalar

To see a full list of Communications System Toolbox blocks that have a new **Rate options** parameter, expand the following section.

Blocks with Rate Options Parameter

- OQPSK Modulator Baseband
- OQPSK Demodulator Baseband
- CPM Modulator Baseband
- CPM Demodulator Baseband
- MSK Modulator Baseband
- MSK Demodulator Baseband
- GMSK Modulator Baseband
- GMSK Demodulator Baseband
- CPFSK Modulator Baseband

- CPFSK Demodulator Baseband
- M-FSK Demodulator Baseband
- M-FSK Modulator Baseband

Compatibility Considerations

Compatibility Considerations

To eliminate this warning, you must upgrade your existing models using the `slupdate` function. The function detects all blocks that have `Inherit from input (this choice will be removed - see release notes)` selected for the **Rate options** 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 **Rate options** parameter to `Enforce single-rate processing`. If the bit is `0` (samples), the function sets the parameter to `Allow multirate processing`.

In a future release, the frame bit and the `Inherit from input (this choice will be removed - see release notes)` option will be removed. At that time, the **Rate options** parameter in models that have not been upgraded will automatically be set to either `Enforce single-rate processing` or `Allow multirate processing`. 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.

R2011b

Version: 5.1
New Features: Yes
Bug Fixes: Yes

New Demos

- The Transceiver Simulation Acceleration demo illustrates simulation acceleration improvements by comparing simulation times using System objects with simulation times using MATLAB functions.
- The Parallel Concatenated Convolutional Coding: Turbo Codes demo now uses the Turbo Encoder and Turbo Decoder blocks and the accompanying MATLAB script uses the `comm.TurboEncoder` and `comm.TurboDecoder` System objects.

Turbo Codes

Communications System Toolbox now supports turbo codes. These error correction codes approach the Shannon limit, resulting in low error rates for transmission schemes with low signal-to-noise ratios. You can implement turbo codes using either MATLAB System objects or Simulink blocks:

- `comm.TurboDecoder`
- `comm.TurboEncoder`
- Turbo Decoder
- Turbo Encoder

USRP2 Migration

Support for the UDP-based USRP2 Transmitter and USRP2 Receiver blocks is being removed in release R2011b. New USRP™ blocks and System objects that work with USRP™ radios using the Universal Hardware Driver™ from Ettus Research™ are now available. These new blocks and objects support buffers with arbitrary frame size. If you have Communications System Toolbox, you can download and use these new blocks and System objects.

GPU System Objects

This release adds new GPU System objects, which use a graphics processing unit (GPU) to procure simulation results more quickly than a CPU. These new objects include:

- comm.gpu.AWGNChannel
- comm.gpu.BlockDeinterleaver
- comm.gpu.BlockInterleaver
- comm.gpu.PSKModulator
- comm.gpu.ViterbiDecoder

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.

Variable-Size Support

The following blocks now support variable-size input and/or output signals:

- APP Decoder
- AWGN Channel (Enter `commvarsize` at the MATLAB command line to access the library containing this implementation of the block)
- CRC-N Generator
- CRC-N Syndrome Detector
- Error Rate Calculation
- General CRC Generator
- General CRC Syndrome Detector
- OSTBC Combiner
- OSTBC Encoder
- Turbo Decoder (Enter `commvarsize` at the MATLAB command line to access the library containing this implementation of the block)
- Turbo Encoder (Enter `commvarsize` at the MATLAB command line to access the library containing this implementation of the block)

The following blocks now support puncturing with variable-size signals:

- Convolutional Encoder
- Viterbi Decoder

The following System objects now support variable-size input and/or output signals:

- comm.APPDecoder
- comm.ConvolutionalEncoder
- comm.CRCDetector
- comm.CRCGenerator
- comm.ErrorRate

- comm.OSTBCCombiner
- comm.OSTBCEncoder
- comm.TurboDecoder
- comm.TurboEncoder
- comm.ViterbiDecoder

System Object Code Generation Support

The following System objects support code generation:

- comm.BarkerCode
- comm.DifferentialDecoder
- comm.DifferentialEncoder
- comm.DiscreteTimeVCO
- comm.HadamardCode
- comm.OVSFCode
- comm.TurboEncoder
- comm.TurboDecoder
- comm.WalshCode

Delayed Reset for Viterbi Decoder

The Viterbi Decoder block and Viterbi Decoder System object now have a delayed reset option. The delay in the reset action allows the block to support HDL code generation. To generate HDL code, you must have an HDL Coder license.

For the Viterbi Decoder block:

- Select **Enable reset input port**
- Select **Delay reset action to next time step**. This parameter only appears when you set the **Operation mode** parameter to `Continuous`.

The Viterbi Decoder block resets its internal state after decoding the incoming data.

For the `comm.ViterbiDecoder` System object

- Set `ResetInputPort` to true
- Set `DelayedResetAction` to true. This property only appears when you set the `ResetInputPort` property to true.
- Set `TerminationMethod` to Continuous

The Viterbi Decoder System object resets its internal state after decoding the incoming data.

System Objects FullPrecisionOverride Property Added

Compatibility Considerations: Yes

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 inputs.

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`

This change affects the following System objects:

- `comm.IntegrateAndDumpFilter`
- `comm.PAMDemodulator`
- `comm.RectangularQAMDemodulator`
- `comm.GeneralQAMDemodulator`

Compatibility Considerations

Compatibility Consideration

All these System objects have their new `FullPrecisionOverride` property set to the default, `true`. If you had set any fixed-point properties to nondefault 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`.

APP Decoder System Object Parameter Change

Compatibility Considerations: Yes

For the `APP Decoder` System object, the `Algorithm` property replaces the `MetricMethod` property. At this time, existing customer code continues to work; however, a warning prompts you to update the code.

Compatibility Considerations

Compatibility Consideration

If you have any existing System object code that uses the `MetricMethod` property, you should use the `sysobjupdate` function to update your code. For more information, type help `sysobjupdate` at the MATLAB command line.

System Object DataType and CustomDataType Properties Changes

Compatibility Considerations: Yes

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

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.

Conversion of System Object Error and Warning Message Identifiers

Compatibility Considerations: Yes

For R2011b, error and warning message identifiers for System objects have changed in Communications System Toolbox software.

Compatibility Considerations

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, 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.

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, 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.

Frame-Based Processing

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. For more information, see “Frame-Based Processing” on page 84.

R2011a

Version: 5.0
New Features: Yes
Bug Fixes: Yes

Product Restructuring

The Communications System Toolbox product replaces two pre-existing products: Communications Blockset and Communications Toolbox. You can access archived documentation for both products on the MathWorks Web site.

LDPC Encoder and Decoder System Objects

This release adds new `comm.LDPCEncoder` and `comm.LDPCDecoder` System objects. These new System objects provide simulation of low-density, parity-check codes.

LDPC GPU Decoder System Object

This release adds a new `comm.gpu.LDPCDecoder` System object, which uses a graphics processing unit (GPU) to decode low-density, parity-check codes. This new System object procures simulation results more quickly than a CPU.

Variable-Size Support

The following blocks now support variable-size input signals:

- M-PSK Modulator Baseband
- QPSK Modulator Baseband
- BPSK Modulator Baseband
- M-PAM Modulator Baseband
- Rectangular QAM Modulator Baseband
- General QAM Modulator Baseband
- M-PSK Demodulator Baseband
- QPSK Demodulator Baseband
- BPSK Demodulator Baseband
- M-PAM Demodulator Baseband
- Rectangular QAM Demodulator Baseband
- General QAM Demodulator Baseband
- Bit to Integer Converter
- Integer to Bit Converter
- Convolutional Encoder
- Viterbi Decoder

The following source blocks can now output variable-size signals:

- Gold Sequence Generator
- Kasami Sequence Generator
- PN Sequence Generator

The following System objects now support variable-size input signals:

- comm.PSKModulator
- comm.QPSKModulator

- comm.BPSKModulator
- comm.PAMModulator
- comm.RectangularQAMModulator
- comm.GeneralQAMModulator
- comm.PSKDemodulator
- comm.QPSKDemodulator
- comm.BPSKDemodulator
- comm.PAMDemodulator
- comm.RectangularQAMDemodulator
- comm.GeneralQAMDemodulator
- comm.IntegerToBit
- comm.BitToInteger

The following System objects now output variable-size signals:

- comm.GoldSequence
- comm.KasamiSequence
- comm.PNSequence

Algorithm Improvements for CRC Blocks

This release introduces a new encoding algorithm for all blocks in the CRC sublibrary residing in the Error Detection and Correction library. In this new implementation, the block processes multiple input bits in one step, resulting in faster processing times. The previous implementation always processed one input bit at each step.

MATLAB Compiler Support for System Objects

The Communications 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:

'Internal rule' System Object Property Values Changed to 'Full precision'

Compatibility Considerations: Yes

To clarify the value of many `DataType` properties, the `'Internal rule'` option has been changed to `'Full precision'`.

Compatibility Considerations

Compatibility Consideration

The objects allow you to enter either `'Internal rule'` or `'Full precision'`. If you enter `'Internal rule'`, that option is stored as `'Full precision'`.

System Object Code Generation Support

The following System objects support code generation:

- comm.PSKTCMMoldulator
- comm.RectangularQAMTCMModulator
- comm.GeneralQAMTCMModulator
- comm.EarlyLateGateTimingSynchronizer
- comm.GardnerTimingSynchronizer
- comm.GMSKTimingSynchronize
- comm.MSKTimingSynchronizer
- comm.MuellerMullerTimingSynchronizer
- comm.KasamiSequence

LDPC Decoder Block Warnings

Compatibility Considerations: Yes

Communications System Toolbox software uses a new implementation of the LDPC Decoder block. If you open a previously existing model that contains the LDPC block, the model generates a warning at the MATLAB command line. Simply resave the model to prevent any subsequent warnings.

Phase/Frequency Offset Block and System Object Change

Compatibility Considerations: Yes

In previous releases, when the frequency offset input signal to the Phase/Frequency Offset block or `comm.PhaseFrequencyOffset` System object was constant, or time-invariant, the block and System object generated the correct output. However, the block and System object produced incorrect results for a time-varying frequency offset input signal. The new implementation generates the correct output for a time-varying frequency offset input signal.

Derepeat Block Changes

The Derepeat block now contains the **Input processing** and **Rate options** parameters. See Frame-Based Processing for more information.

Version 2, 2.5, and 3.0 Obsolete Blocks Removed

Compatibility Considerations: Yes

All the obsolete block libraries associated with Communications Blockset version 2 Release 12, version 2.5 Release 13, and version 3.0 Release 14 have been removed from this product. The removal includes the following libraries:

- `commanabbnd2`
- `commcontsrc2`
- `commdigpbndam2`
- `commdigpbndcpm2`
- `commdigpbndfm2`
- `commdigpbndpm2`
- `comminteg2`
- `commanapbnd2`
- `commchan2`
- `commdigbbndam2`
- `commdigbbndpm2`

Compatibility Considerations

Compatibility Considerations

Communications System Toolbox software does not support any of the blocks from Release 12 and Release 13. The Communications System Toolbox block libraries provide some of the same functionality in the form of upgraded blocks.

System Objects Input and Property Warnings Changed to Errors

Compatibility Considerations: Yes

When a System object is locked (for example, 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

Compatibility Consideration

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.

Frame-Based Processing

Compatibility Considerations: Yes

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. Communications 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 transition to the new paradigm of frame-based processing, many blocks have received new parameters. The following sections provide more detailed information about the specific Communications 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 86
- “Multirate Processing Parameter Changes” on page 88
- “Sample-Based Row Vector Processing Changes” on page 90

Compatibility Considerations

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 R2010b 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 the **Input processing** parameter, the parameter will be set to `Inherited (this choice will be removed - see release notes)` when you load your model. 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 R2010b frame-based processing changes, see the following Compatibility Considerations sections.

Blocks with a New Input Processing Parameter

Some Communications 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 received a new **Input processing** parameter. You can select `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, expand the following list.

Blocks with New Input Processing Parameter

- Derepeat
- Gaussian Filter
- Windowed Integrator
- AWGN Channel (with only two options)

Compatibility Considerations

Compatibility Considerations

When you load an existing model R2010b, 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 R2010b, 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 for the Gaussian Filter or Windowed Integrator, 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.

AWGN Channel Block Changes

The AWGN Channel block uses the new method of "Frame-Based Processing" on page 84. In previous releases, the frame status of the input signal determined how the AWGN Channel block processed the signal. In R2010b, the default behavior of the AWGN Channel block is to always perform frame-based processing.

Unless you specify otherwise, the block now treats each column of the input signal as an individual channel, regardless of its frame status. To enable the behavior change in the AWGN Channel block while still allowing for backward compatibility, an **Input processing** parameter has been added. This parameter will be removed in a future release, at which point the block will always perform frame-based processing.

Compatibility Considerations

Compatibility Considerations

The **Input processing** parameter will be removed in a future release. At that point in time, the AWGN Channel block will always perform frame-based processing.

You can use the `slupdate` function to upgrade your existing models that contain an AWGN Channel block. The function detects all AWGN Channel blocks in your model and, if you allow it to, performs the following actions:

- If the input to the block is an *M*-by-1 or unoriented sample-based signal, the `slupdate` function performs three actions. First, a Transpose block is placed in front of the AWGN Channel 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. The `slupdate` function also sets the **Input processing** parameter to `Columns as channels (frame based)`. This setting ensures that your model will continue to produce the same results when the **Input processing** parameter is removed in a future release. The `slupdate` function also adds a Transpose block after the AWGN channel block in your model for an *M*-by-1 sample-based input and a Reshape block for unoriented inputs. By converting the row vector output of the AWGN channel to the input dimension, the model continues to behave as in prior releases.
- If the input to the block is *not* an *M*-by-1 or unoriented sample-based signal, the `slupdate` function sets the **Input processing** parameter to `Columns as channels (frame based)`. 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 **Input processing** parameter is removed in a future release.

Multirate Processing Parameter Changes

In R2010a and earlier releases, many Communications System Toolbox blocks that supported multirate processing had a **Framing** parameter. This parameter allowed you to specify whether the block should `Maintain input frame size` or `Maintain input frame rate` when processing the input signal. Beginning in R2010b, a new **Rate options** parameter replaced the

Framing parameter. The **Rate options** parameter allows you to specify whether the block should `Enforce single-rate processing` or `Allow multirate processing`.

Some blocks that supported multirate processing in R2010a and earlier releases did not have a **Framing** parameter. These blocks used the frame status of the input signal to determine whether they performed single-rate or multirate processing. Because of the upcoming frame-based processing changes, signals will no longer carry their frame status. Thus, multirate blocks can no longer rely on the frame status of the input signal to determine whether they perform single-rate or multirate processing. You must now specify a value for the **Rate options** parameter on the block dialog box.

To see a full list of blocks that have a new **Rate options** parameter, expand the following section.

Multirate Blocks with a New Rate Options Parameter

- Raised Cosine Receive Filter
- Raised Cosine Transmit Filter
- Ideal Rectangular Pulse Filter
- OQPSK Modulator Baseband
- OQPSK Demodulator Baseband
- CPM Modulator Baseband
- CPM Demodulator Baseband
- MSK Modulator Baseband
- MSK Demodulator Baseband
- GMSK Modulator Baseband
- GMSK Demodulator Baseband
- CPFSK Modulator Baseband
- CPFSK Demodulator Baseband
- M-FSK Demodulator Baseband
- M-FSK Modulator Baseband

- Derepeat

Sample-Based Row Vector Processing Changes

The following blocks do not process sample-based row vectors:

- APP Decoder
- Convolutional Encoder
- Viterbi Decoder
- Algebraic Deinterleaver
- Algebraic Interleaver
- General Block Deinterleaver
- General Block Interleaver
- Matrix Deinterleaver
- Matrix Helical Scan Deinterleaver
- Matrix Helical Scan Interleaver
- Matrix Interleaver
- Random Deinterleaver
- Random Interleaver
- M-PAM Modulator Baseband
- Rectangular QAM Modulator Baseband
- DQPSK Modulator Baseband
- M-DPSK Modulator Baseband
- M-PSK Modulator Baseband
- OQPSK Modulator Baseband
- QPSK Modulator Baseband
- M-FSK Modulator Baseband
- CPFSK Modulator Baseband
- CPM Modulator Baseband

- Insert Zero
- Puncture
- Bit to Integer Converter
- Integer to Bit Converter

Compatibility Considerations

Compatibility Considerations

Using existing models that contain these blocks to process sample-based row vectors generates an error message.

CMA Equalizer Changes

The CMA Equalizer block now handles input signals like the other equalizer blocks in the Communications Blockset library. Therefore, the block no longer accepts scalar input signals in symbol-spaced mode.

Differential Encoder Changes

The Differential Encoder block supports scalar-valued and column vector input signals. It does not support frame-based or sample-based row vectors.

Find Delay and Align Signal Block Changes

The **Correlation window length** parameter specifies the number of samples the block uses to calculate the cross-correlation of two signals. You must specify a window lengths of at least `2` for the cross-correlation calculations. If you set the **Correlation window length** parameter to `1`, the block generates an error message. The following blocks contain the **Correlation window length** parameter:

- Find Delay
- Align Signals

New Demos

This release contains the following new demos:

- Parallel Concatenated Convolutional Coding: Turbo Codes
- Go-Back-N ARQ with PHY Layer
- Adaptive MIMO System with OSTBC
- CORDIC-Based QPSK Carrier Synchronization
- DVB-S.2 Link, Including LDPC Coding
- DVB-S.2 System Simulation Using a GPU-Based LDPC Decoder System Object