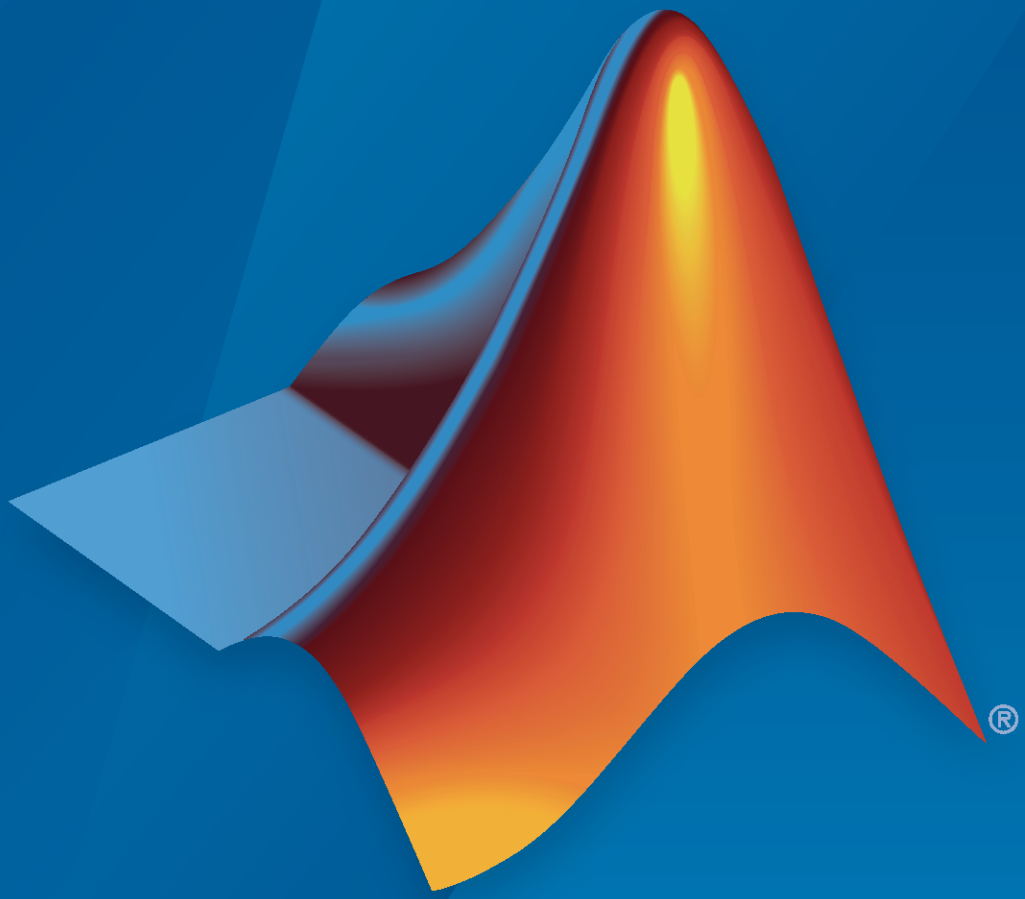
Satellite Communications Toolbox

Reference



MATLAB®

R2021a

MathWorks®

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.
1 Apple Hill Drive
Natick, MA 01760-2098

Satellite Communications Toolbox Reference

© COPYRIGHT 2021 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.

Revision History

March 2021	Online only	New for Version 1.0 (Release 2021a)

Contents

1 Apps

2 Functions

3 Objects

4 System Objects

Apps

Satellite Link Budget Analyzer

Analyze link budgets for satellite communications

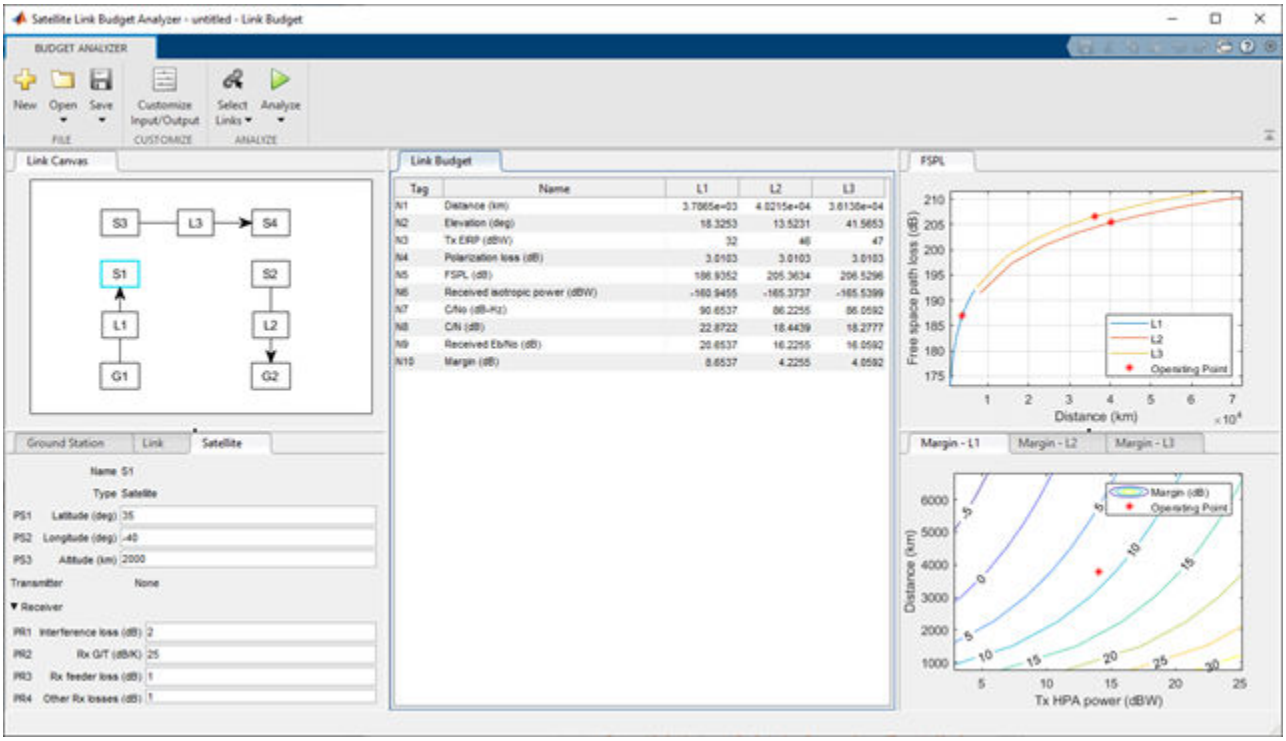
Description

The **Satellite Link Budget Analyzer** app enables you to analyze link budgets for satellite communications.

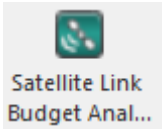
Using the app, you can:

- Analyze link budgets by specifying inputs properties related to
  - Location, transmitter, and receiver characteristics for satellites and ground stations
  - Atmospheric conditions for links
- Design a satellite communications link to meet a minimum link margin requirement
- Have insight into intermediate link budget computations
- Calculate, compare, and visualize results across a sweep of multiple parametrized design constraints

For more information, see “Get Started with Satellite Link Budget Analyzer App”.



Open the Satellite Link Budget Analyzer App

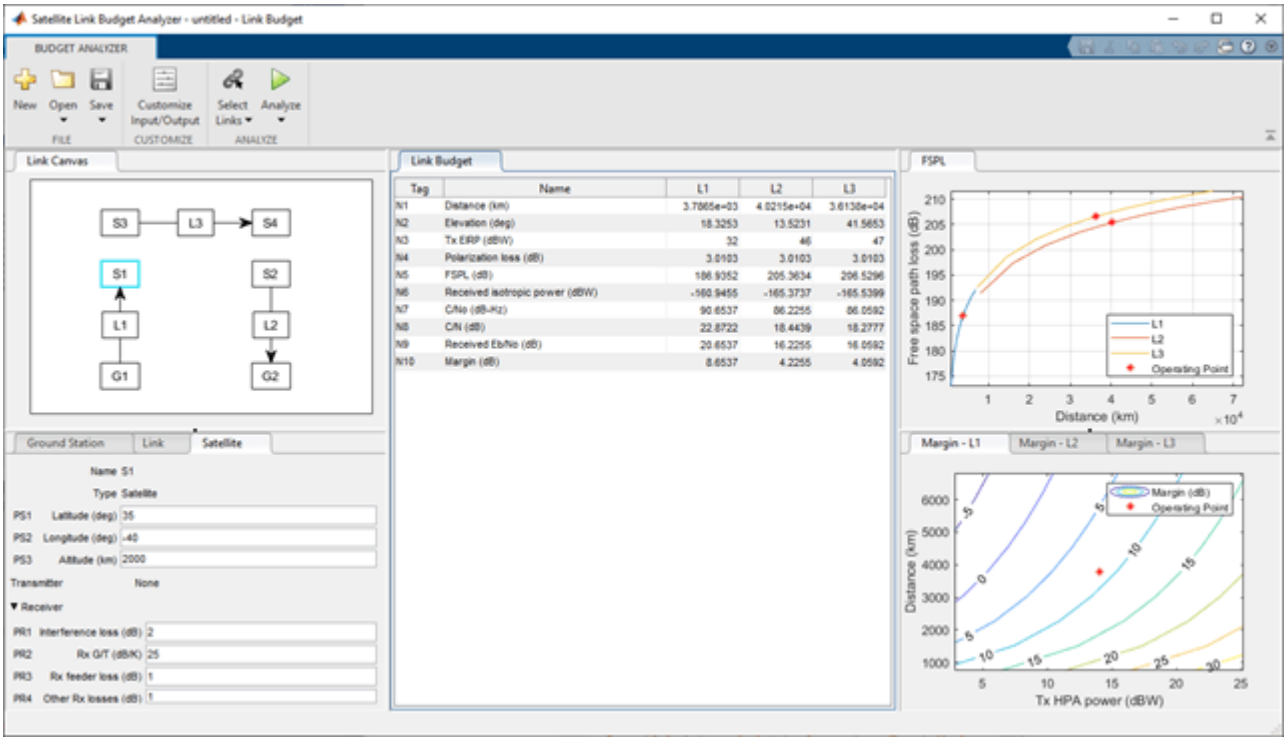
MATLAB® Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon. 

MATLAB Command Prompt: Enter `satelliteLinkBudgetAnalyzer`.

Examples

Show Default Satellite Link Budget App Configuration

This example shows the default configuration that appears when you open the **Satellite Link Budget Analyzer** app. The displayed results and plots analyze the default satellite communications link.



The upper left area of the app window shows the **Link Canvas** tab on top with this default configuration:

- Link L1 is an uplink connecting ground station G1 to satellite S1
- Link L3 is a crosslink connecting satellite S3 to satellite S4
- Link L2 is a downlink connecting satellite S2 to ground station G2

The lower left area of the app window shows the **Ground Station**, **Link**, and **Satellite** tabs. In these tabs you can adjust property settings for each entity in the configured links. To view or adjust the properties settings of an entity, bring it into focus by selecting it in the **Link Canvas** tab.

The center area of the app window shows the computed link budget results in the **Link Budget** tab.

The right area of the app window shows these plots:

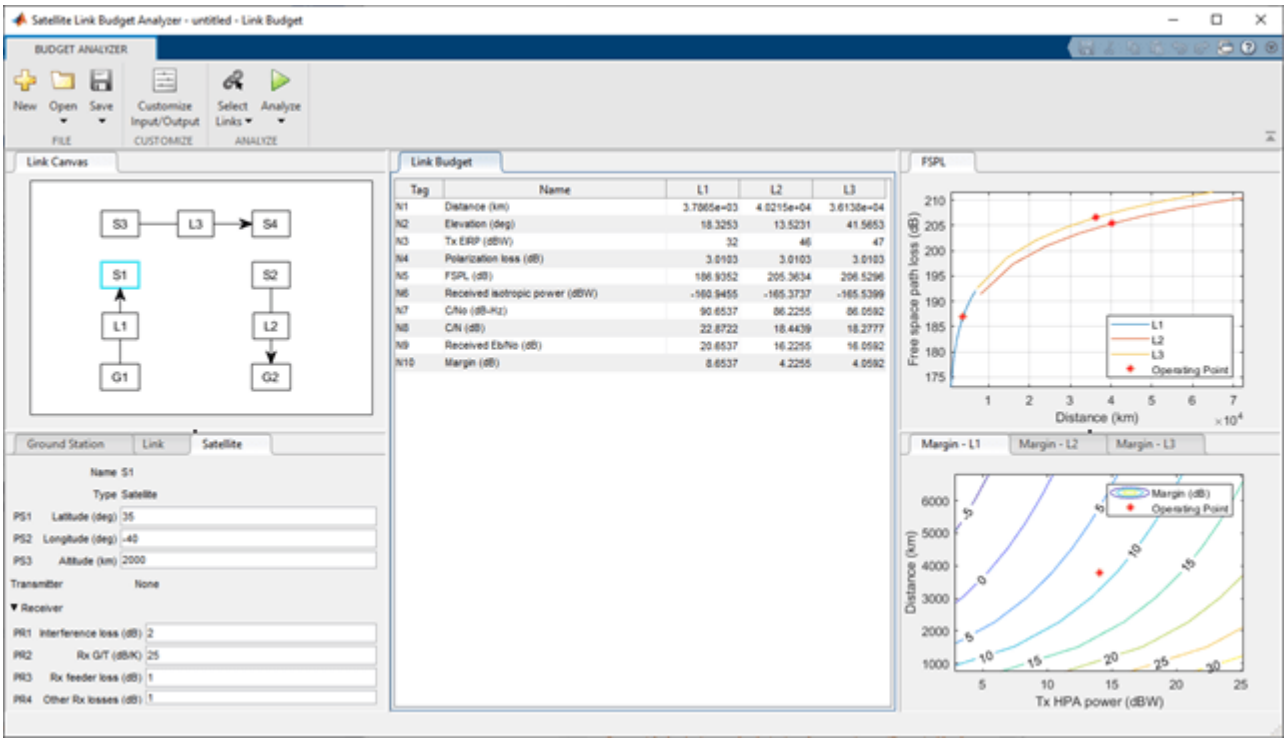
- Free-space path loss for links L1, L2, and L3 in the upper-right area.
- Link margins for links L1, L2, and L3 in separate tabbed plots in the lower-right area.

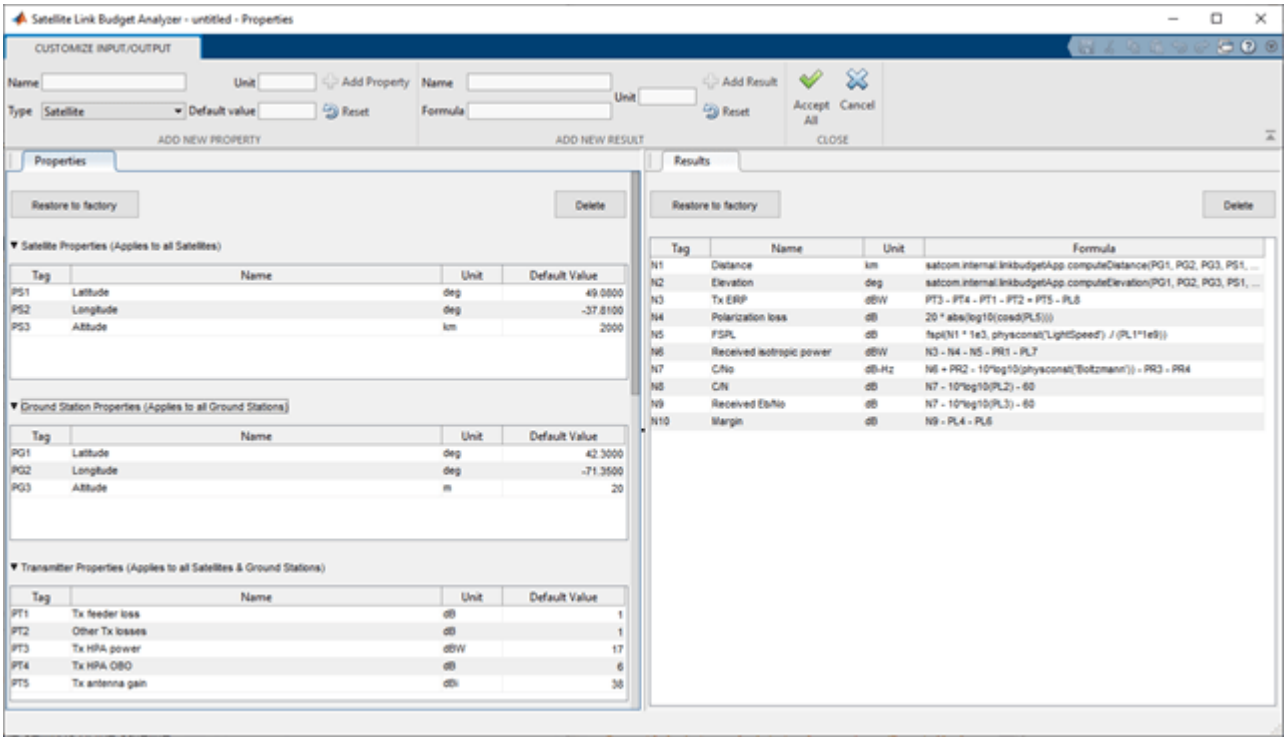
Analyze Satellite Availability in P.618 Propagation Model

Configure the **Satellite Link Budget Analyzer** app to analyze satellite communications link availability through the propagation loss model defined in Recommendation ITU-R P.618-13. Load the `AvailabilityAnalysis-P618.mat` saved session to configure the app for the P.618 availability analysis. This example uses the model and API provided by the P.618 Recommendation. For details on the P.618 propagation loss model, see “Earth-Space Propagation Losses”. Before configuring the app for the P-618 availability analysis, download and unpack the ITURDigitalMaps.tar.gz file to a directory that is on the MATLAB path.

Configure Satellite Link Budget Analyzer App

Open the **Satellite Link Budget Analyzer** app. These figures show the default configuration on the **Budget Analyzer** and **Customize Input/Output** tabs.



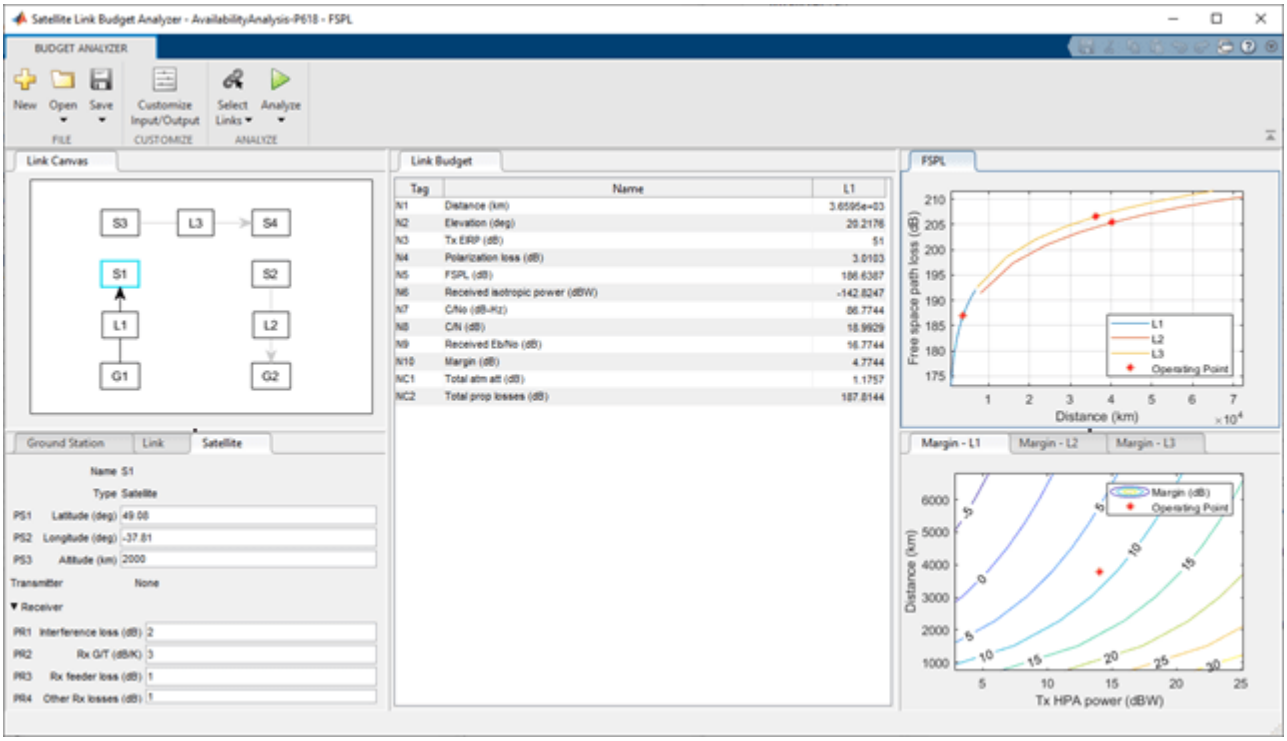


Click **Open** on the **Budget Analyzer** tab, and then open the `AvailabilityAnalysis-P618.mat` saved session file from the working directory of this example.

Opening the saved `AvailabilityAnalysis-P618.mat` session updates the app configuration by adding new properties, adjusting formulas for results, and simplifying the workflow by removing links and computations not relevant to this analysis scenario.

After the saved session loads, the **Budget Analyzer** tab shows the configuration updates. On the **Budget Analyzer** tab,

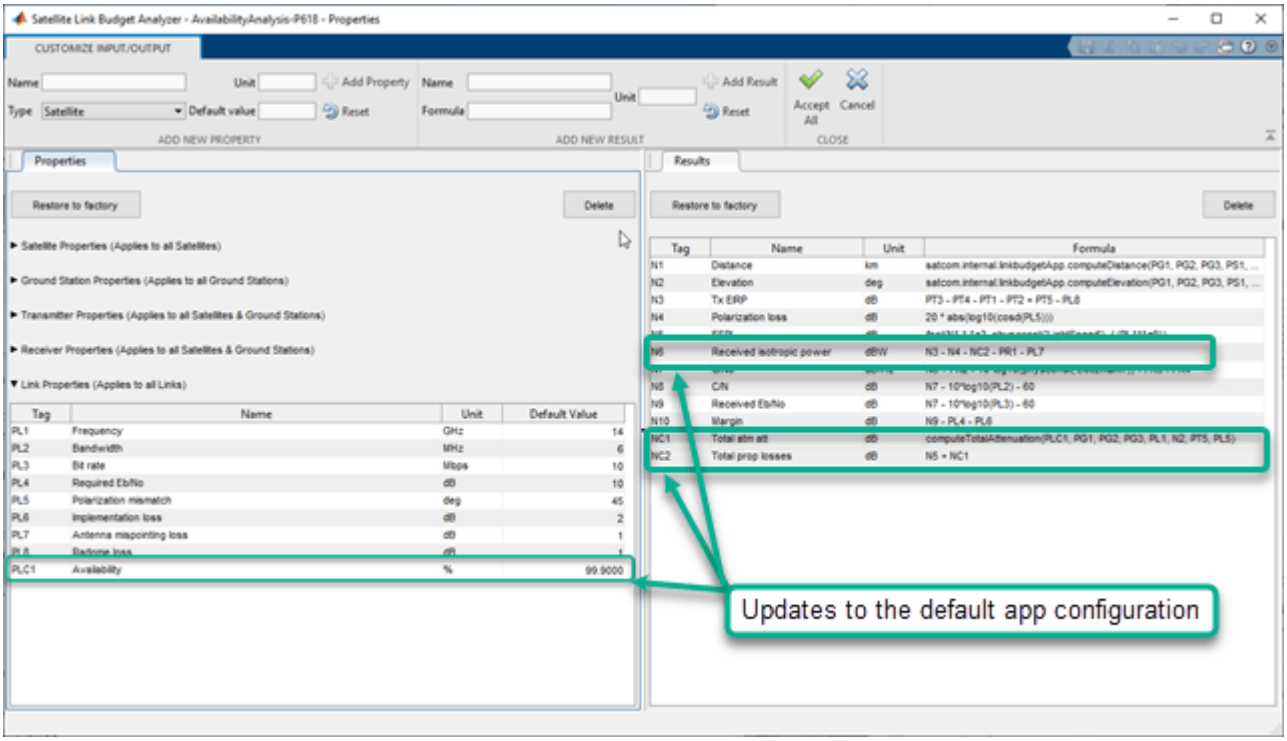
- **Analyze** shows **Auto Analyze** not selected.
- **Select Links** shows that only link **L1** is selected. The analysis in this example requires only the uplink path (link **L1**).Computation of results for links **L2** and **L3** are turned off.



After the saved session loads, the **Customize Input/Output** tab shows the configuration updates.

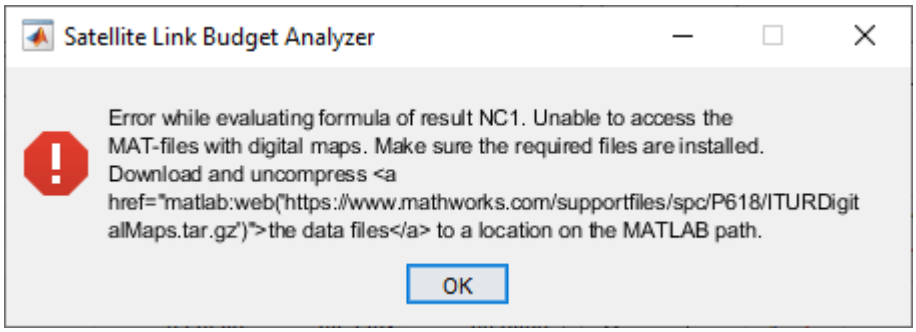
- A property is added to the **Link Properties** section of the **Properties** tab to specify the desired availability percentage (**Availability**). The value is set to 99.9%.
- Results are added on the **Results** tab to compute the total atmospheric attenuation (`Total atm att`) and total propagation losses (`Total prop losses`) per the P.618 propagation model. To compute the total atmospheric attenuation, the app runs the computeTotalAttenuation.m function using properties configured in the app. In the formula for the `Total atm att` property on the **Results** tab you can see the input parameters as identified by property tag names (`PLC1`, `PG1`, `PG2`, `PG3`, `PL1`, `N2`, `PT5`, and `PL5`).
- The formula for the received isotropic power (Tag: `N6`) on the **Results** tab is changed to use `NC2` instead of `N5`.

This figure shows updates to the configuration in the **Properties** and **Results** tabs.



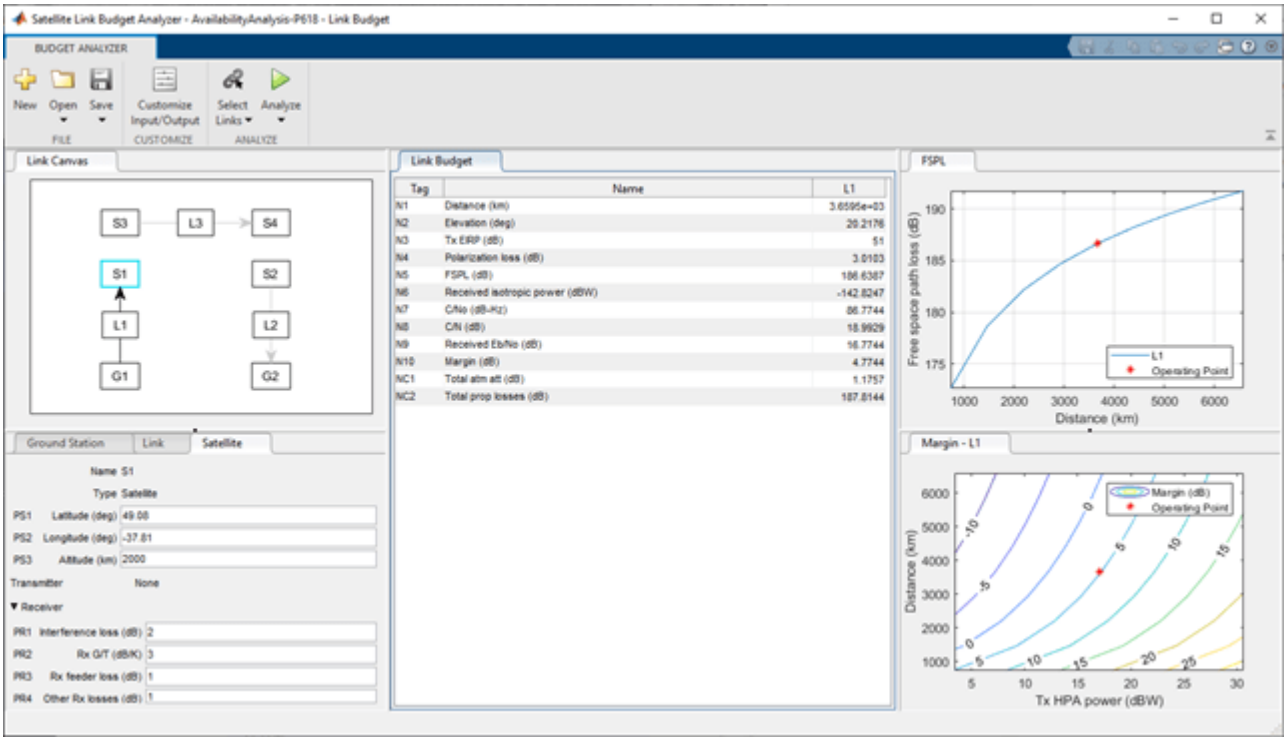
Analyze Link Availability

Before analyzing the P.618 scenario, you must download the digital maps file. If you have not downloaded and unpacked the digital maps file, this dialog box will appear. Follow the instructions in the dialog.



After you have downloaded and uncompressed the digital maps file, click **Analyze** to compute the link budget for the configured P.618 scenario.

The app runs the P.618 scenario to analyze the 99.9% annual availability and then updates the link budget and plotted results. The 99.9% annual availability is satisfied with approximately 4.8 dB of margin for the configured satellite communications link.

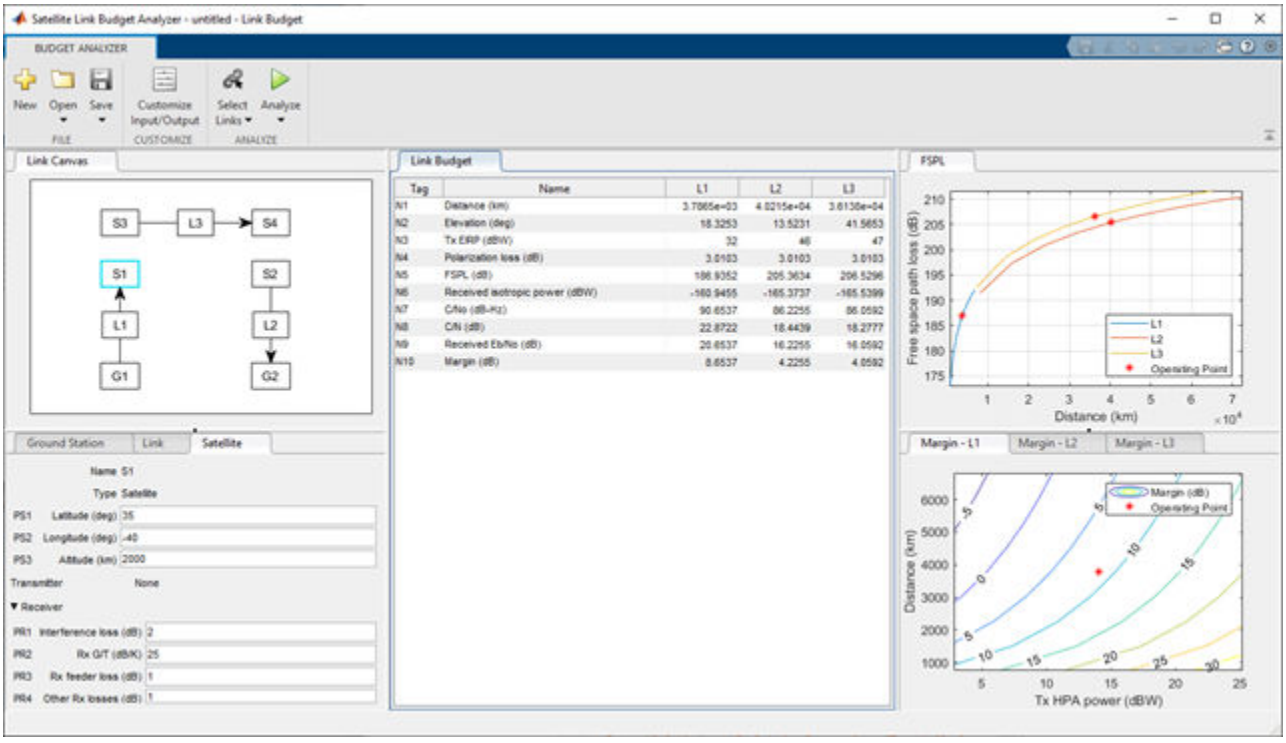


Parameters

BUDGET ANALYZER — Link budget configuration

tab

This figure shows the **BUDGET ANALYZER** tab with the factory default configuration.



Use the **Ground Station**, **Link**, and **Satellite** tabs to adjust property settings for the link budget entities shown in the **Link Canvas** tab.

`Ground Station` — Ground station location, transmitter, and receiver settings
tab

Select the **Ground Station** tab to set the location, transmitter, and receiver settings for the ground station highlighted in the **Link Canvas** tab. For information about customizing satellite, ground station, transmitter, receiver, and link properties, and the link budget result computations, see `CUSTOMIZE INPUT/OUTPUT`.

`Satellite` — Satellite location, transmitter, and receiver settings
tab

Select the **Satellite** tab to set the location, transmitter, and receiver settings for the satellite highlighted in the **Link Canvas** tab. For information about customizing satellite, ground station, transmitter, receiver, and link properties, and the link budget result computations, see `CUSTOMIZE INPUT/OUTPUT`.

`Link` — Link characteristics
tab

Select the **Link** tab to set link characteristics for the link highlighted in the **Link Canvas** tab. For information about customizing satellite, ground station, transmitter, receiver, and link properties, and the link budget result computations, see `CUSTOMIZE INPUT/OUTPUT`.

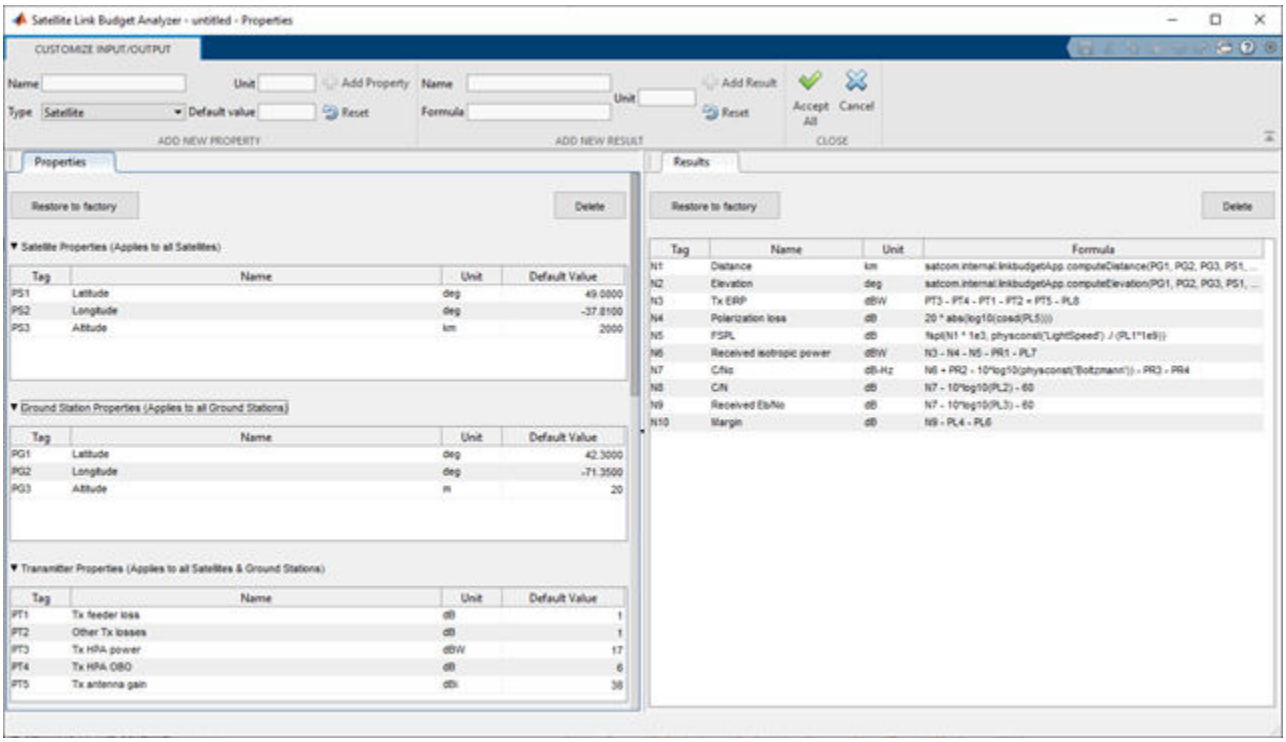
`Customize Input/Output` — Customize input properties and computations used for output
tab

To view or customize input properties and computations used for output, on the **BUDGET ANALYZER** tab, click **Customize Input/Output** to switch to the **CUSTOMIZE INPUT/OUTPUT** tab. In the **CUSTOMIZE INPUT/OUTPUT** tab, you can

- Change settings of the satellite, ground station, transmitter, receiver, and link properties from the factory default inputs
- Add and delete satellite, ground station, transmitter, receiver, and link input properties
- Add, delete, and modify formulas used to compute link budget output results

`CUSTOMIZE INPUT/OUTPUT` — Customize link budget computations tab

This figure show the **CUSTOMIZE INPUT/OUTPUT** tab with the factory default configuration.



In the **CUSTOMIZE INPUT/OUTPUT** tab, you can

- Use the **Properties** tab to change settings of the satellite, ground station, transmitter, receiver, and link properties from the factory default inputs. You can also add and delete satellite, ground station, transmitter, receiver, and link input properties. On the **Properties** tab you can use the **Restore to factory** button to load the factory default property configuration in the current app session.
- Use the **Results** tab to add, delete, and modify formulas used to compute link budget output results. On the **Results** tab you can use the **Restore to factory** button to load the factory default results configuration in the current app session.

Programmatic Use

`satelliteLinkBudgetAnalyzer` opens the **Satellite Link Budget Analyzer** app.

See Also

Functions
`fspl`

Objects
`satelliteScenario`

Topics
“Get Started with Satellite Link Budget Analyzer App”

Introduced in R2021a

Functions

ccsdsRSEncode

CCSDS-compliant Reed-Solomon encoder

Syntax

```
code = ccsdsRSEncode(msg,k)
code = ccsdsRSEncode(msg,k,i)
code = ccsdsRSEncode(msg,k,i,s)
```

Description

`code = ccsdsRSEncode(msg,k)` encodes the message in `msg` by using a (255, `k`) Reed-Solomon (RS) encoder, as defined in Consultative Committee for Space Data Systems (CCSDS) 131.0-B-3 Section 4 [1]. `k` is the message length. `code` is in dual basis form, as the function assumes that the input to the CCSDS RS encoder is in dual basis form. For more details on dual basis representation, see CCSDS 131.0-B-3 Section 4.4.2 [1].

For a description of CCSDS RS code construction, see “CCSDS RS Code Construction” on page 2-5.

`code = ccsdsRSEncode(msg,k,i)` specifies the interleaving depth, `i`. `msg` consists of `i` RS message symbols of length `k`.

`code = ccsdsRSEncode(msg,k,i,s)` encodes the shortened input message of length `s` with interleaving depth `i`.

Examples

Encode Message Using Full-Length CCSDS RS Encoder

Encode a message using a Consultative Committee for Space Data Systems (CCSDS) Reed-Solomon (RS) encoder.

Specify the message length, `k`, and the interleaving depth, `i`.

```
k = 239;
i = 3;
```

Generate a column vector of random message symbols. The length of the message is product of message length, `k`, and interleaving depth, `i`.

```
msg = randi([0 255],k*i,1);
size(msg)
```

```
ans = 1×2

   717     1
```

Encode the message by using CCSDS RS encoder.

```
code = ccsdsRSEncode(msg,k,i);
```

Verify that the length of the encoded codeword is 255 times the value of the interleaving depth.

```
size(code)
```

```
ans = 1×2

   765     1
```

Encode Shortened Message Using CCSDS RS Encoder

Encode a message using a Consultative Committee for Space Data Systems (CCSDS) Reed-Solomon (RS) encoder with message shortening.

Specify the message length, `k`, interleaving depth, `i`, and the shortened message length, `s`.

```
k = 223;
i = 2;
s = 146;
```

Generate a column vector of random message bits. The length for the shortened message bits is eight times the product of shortened message length, `s`, and the interleaving depth, `i`.

```
msg = logical(randi([0 1],s*i*8,1));
```

Encode the shortened message by using a CCSDS RS encoder.

```
code = ccsdsRSEncode(msg,k,i,s);
```

Verify that the length of the encoded codeword is equal to (8*`i`*(255 – `k` + `s`).

```
size(code)
```

```
ans = 1×2

        2848           1
```

Input Arguments

`msg` — Input message

column vector of logical bits | column vector of integers in the range [0, 255]

Input message, specified as a column vector of logical bits or a column vector of integers in the range [0, 255]. The size of the column vector depends on the data type of the input message.

Input Message Type	Size of `msg`	
	Data Type of `msg` Is `logical`	Data Type of `msg` Is `uint8` or `double`
Full-length input message	8*k	k

Input Message Type	Size of msg	
	Data Type of msg Is logical	Data Type of msg Is uint8 or double
Interleaved input message	8*k*i	k*i
Shortened input message	8*s*i	s*i

Data Types: double | uint8 | logical

k — Message length
223 | 239

Message length, specified as 223 or 239.

Data Types: double

i — Interleaving depth
1 (default) | 2 | 3 | 4 | 5 | 8

Interleaving depth, specified as 1, 2, 3, 4, 5, or 8. The default value, 1, corresponds to no interleaving.

msg consists of i RS message symbols of length k.

Data Types: double

s — Shortened message length
k (default) | integer in the range [1, k]

Shortened message length, specified as an integer in the range [1, k].

Data Types: double

Output Arguments

code — CCSDS RS encoded message
column vector

CCSDS RS encoded message, returned as a column vector. The data type of code is same as that of the input message, msg. The size of the column vector depends on the data type of the input message.

Input Message Type	Size of code	
	Data Type of msg Is logical	Data Type of msg Is uint8 or double
Full length input message	8*255	255
Interleaved input message	8*255*i	255*i
Shortened input message	8*i*(255 – k + s)	i*(255 – k + s)

More About

CCSDS RS Code Construction

CCSDS RS codes are powerful burst error-correcting codes used as forward error-correcting (FEC) codes.

The CCSDS RS encoder accepts full-length or shortened messages.

Construction of Full-Length Message CCSDS RS Codes

For full-length input messages the input column vector length is a product of the interleaving depth (i) and the message length (k).

Encoding in CCSDS RS codes is done row-wise. The encoding results in an i-by-n vector that includes parity bits added to the end of each row. n is the codeword length, which is fixed to 255 symbols according to CCSDS 131.0-B-3 Section 4 [1].

Construction of Shortened Message CCSDS RS Codes

For shortened input messages, the input column vector length is a product of the interleaving depth (i) and the shortened message length (s). The shortened message vector prepends padding the beginning of the message vector with zeros. The resulting vector is an i-by-k vector.

Encoding in CCSDS RS codes is done row-wise. The encoding results in an i-by-n vector that includes parity bits added to the end of each row.

References

[1] TM Synchronization and Channel Coding. *Recommendation for Space Data System Standards*. CCSDS 131.0-B-3. Blue Book. Issue 3. Washington, D.C.: CCSDS, September 2017.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`ccsdsRSDecode`

Objects

`ccsdsTMWaveformGenerator` | `comm.RSEncoder`

Introduced in R2021a

ccsdsRSDecode

CCSDS-complaint Reed-Solomon decoder

Syntax

```
[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k)
[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k,i)
[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k,i,s)
```

Description

`[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k)` decode the received signal in `code` by using a (255, `k`) Reed-Solomon (RS) decoder with the generator polynomial, as defined in the Consultative Committee for Space Data Systems (CCSDS) 131.0-B-3 Section 4 [1]. `k` is the number of symbols in the decoded message. The function returns the decoded message `code`, `decoded`, the number of corrected errors, `cnumerr`, and the corrected version of `code`, `ccode`.

For a description of CCSDS RS code decoding, see “CCSDS RS Code Decoding” on page 2-9.

`[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k,i)` specifies the interleaving depth, `i`. `code` consists of `i` RS codewords of length 255 bytes.

`[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k,i,s)` specifies the shortened message length, `s`.

Examples

Encode and Decode Full-length CCSDS RS Encoded Message

Generate a full-length encoded Reed-Solomon (RS) codeword, introduce random errors, and decode the result using a Consultative Committee for Space Data Systems (CCSDS) RS decoder.

Generate a random message of length `k`.

```
k = 223;
msg = randi([0 255],k,1);
```

Encode the message by using a CCSDS RS encoder.

```
code = ccsdsRSEncode(msg,k);
```

Generate 15 random error symbols and 15 unique random locations to insert these errors.

```
err = randi([1 255],15,1);
errLoc = randperm(255,15);
errVec = zeros(255,1);
errVec(errLoc) = err;
```

Introduce error symbols in the encoded message.

```
rxBytes = bitxor(code,errVec);
```

Decode the encoded symbols introduced with errors by using CCSDS RS decoder.

```
[decoded,v,ccode] = ccsdsRSDecode(rxBytes, k);
```

Display the number of corrected errors.

```
disp(v)
```

```
    15
```

Decode CCSDS RS Codeword with Burst Errors

Generate an full-length encoded Reed-Solomon (RS) codeword, introduce burst of erros, and decode the result using a Consultative Committee for Space Data Systems (CCSDS) RS decoder.

Specify the message length `k` and interleaving depth, `i`.

```
k = 239;
i = 5;
```

Generate a column vector of random message bits. Encode the shortened message by using a CCSDS RS encoder.

```
msg = randi([0 255],k*i,1);
code = ccsdsRSEncode(msg,k,i);
```

Generate 30 random error symbols.

```
err = randi([1 255],30,1);
errVec = zeros(255*i,1);
```

Introduce burst errors from location 52 to 81.

```
errVec(52:81) = err;
rxBytes = bitxor(code,errVec);
```

Decode the encoded symbols introduced with burst errors by using a CCSDS RS decoder.

```
[decoded,v,ccode] = ccsdsRSDecode(rxBytes,k,i);
```

Display the number of corrected errors.

```
disp(v)
```

```
    30
```

Input Arguments

code — Encoded message

column vector of integers in the range [0, 255]

Encoded message, specified as a column vector of integers in the range [0, 255].

The elements and the size of the column vector depends on the data type of the input message.

- For a logical data type, each element in the vector is either 0 or 1.
- For a uint8 or double data type, each element is an integer symbol in GF(2^m), in the range [0, 255]. *m* is the number of bits in each symbol.

Input Message Type	Size of code	
	Data Type of code Is logical	Data Type of code Is uint8 or double
Full length input message	8*255	255
Interleaved input message	8*255*i	255*i
Shortened input message	8*i*(255 – k + s)	i*(255 – k + s)

Data Types: `double` | `uint8` | `logical`

k — Number of symbols in decoded message
223 | 239

Number of symbols in the decoded message, specified as 223 or 239.

Data Types: `double`

i — Interleaving depth
1 (default) | 2 | 3 | 4 | 5 | 8

Interleaving depth, specified as 1, 2, 3, 4, 5, or 8. The default value, 1, corresponds to no interleaving.

`code` consists of `i` RS codewords of length 255 bytes.

Data Types: `double`

s — Shortened message length
k (default) | integer in the range [1, k]

Shortened message length, specified as an integer in the range [1, k].

Data Types: `double`

Output Arguments

decoded — Decoded message
column vector

Decoded message, returned as a column vector. Each element represents decoding the corresponding element in input `code`. The data type of `decoded` is the same as that of `code`.

The size of the column vector depends on the data type of `code`.

Input Message Type	Size of decoded	
	Data Type of code Is logical	Data Type of code Is uint8 or double
Full length input message	8*k	k
Interleaved input message	8*k*i	k*i
Shortened input message	8*s*i	s*i

When the value of output `cnumerr` is `–1`, `decoded` is equal to the first `k` elements of `code`.

`cnumerr` — Number of corrected errors
integer in the range [-1, (*n* – k) / 2]

Number of corrected errors, returned as an integer in the range [-1, (*n* – k) / 2], where *n* is the codeword length. The value of *n* is set to `255` according to CCSDS 131.0-B-3 Section 4 [1].

A value of `–1` in `cnumerr` indicates the failure of the decoder to correct the errors.

`ccode` — Corrected version of code
column vector

Corrected version of code, returned as a column vector. The length of `ccode` is same as the length of `code`. The data type of `ccode` is the same as that of `code`.

When the value of output `cnumerr` is `–1`, `ccode` is equal to `code`.

More About

CCSDS RS Code Decoding

CCSDS RS codes are powerful burst error-correcting codes. These are most commonly used as forward error-correcting (FEC) codes, as they detects and correct errors on the symbol level.

Decoding Full-Length Message CCSDS RS Codes

Like encoding, decoding of CCSDS RS codes is also done row-wise. The input vector length is a product of interleaving depth (*i*) and codeword length (*n*). *n* is fixed to 255 symbols according to CCSDS 131.0-B-3 Section 4 [1]. The input vector is composed of message and parity symbols.

Decoding Shortened Message CCSDS RS Codes

Like encoding, the decoding of CCSDS RS codes is also done row-wise. The input vector length is a product of the interleaving depth (*i*) and the value calculated by *n*-*k*+*s*. The input vector is composed of shortened message and parity symbols.

References

[1] TM Synchronization and Channel Coding. *Recommendation for Space Data System Standards*. CCSDS 131.0-B-3. Blue Book. Issue 3. Washington, D.C.: CCSDS, September 2017.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`ccsdsRSEncode`

Objects

`ccsdsTMWaveformGenerator` | `comm.RSDecoder`

Introduced in R2021a

dvbs2BitRecover

Recover bits for DVB-S2 PL frames

Syntax

```
[BITS,NUMFRAMESLOST] = dvbs2BitRecover(RXFRAME,NVAR)
[BITS,NUMFRAMESLOST,PKTCRCSTATUS] = dvbs2BitRecover(RXFRAME,NVAR)
[BITS,NUMFRAMESLOST] = dvbs2BitRecover(RXFRAME,NVAR,EARLYTERM)
```

Description

`[BITS,NUMFRAMESLOST] = dvbs2BitRecover(RXFRAME,NVAR)` recovers user packets (UPs) or a continuous data stream, `BITS`, and the number of lost baseband frames, `NUMFRAMESLOST`. Input `RXFRAME` is the received complex in-phase quadrature (IQ) symbols in the form of physical layer (PL) frames of a Digital Video Broadcasting Satellite Second Generation (DVB-S2) transmission. Input `NVAR` is the noise variance estimate, used to calculate soft bits.

`[BITS,NUMFRAMESLOST,PKTCRCSTATUS] = dvbs2BitRecover(RXFRAME,NVAR)` also returns the UP cyclic redundancy check (CRC) status.

`[BITS,NUMFRAMESLOST] = dvbs2BitRecover(RXFRAME,NVAR,EARLYTERM)` uses low-density parity-check (LDPC) decoding termination criterion, `EARLYTERM`, to recover data bits, `BITS`.

Examples

Recover Data Bits from Transport Stream DVB-S2 Transmission

Recover user packets (UPs) for multiple physical layer (PL) frames in a single transport stream Digital Video Broadcasting Satellite Second Generation (DVB-S2) transmission.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream. Create a DVB-S2 System object.

```
nFrames = 2;
s2WaveGen = dvbs2WaveformGenerator;
```

Create the bit vector of information bits, `data`, of concatenated TS UPs.

```
syncBits = [0 1 0 0 0 1 1 1]';    % Sync byte for TS packet is 47 Hex
pktLen = 1496;                    % UP length without sync bits is 1496
```

```
numPkts = s2WaveGen.MinNumPackets*nFrames;
txRawPkts = randi([0 1],pktLen,numPkts);
txPkts = [repmat(syncBits,1,numPkts); txRawPkts];
data = txPkts(:);
```

Generate the DVB-S2 time-domain waveform using the input information bits. Flush the transmit filter to handle the filter delay and recover the complete last frame.

```
txWaveform = [s2WaveGen(data); flushFilter(s2WaveGen)];
```

Add additive white Gaussian noise (AWGN) to the generated waveform.

```
sps = s2WaveGen.SamplesPerSymbol;
EsNodB = 1;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn(txWaveform,snrdB,'measured');
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
      'RolloffFactor',s2WaveGen.RolloffFactor, ...
       'InputSamplesPerSymbol',sps,...
       'DecimationFactor',sps);
s = coeffs(rxFilter);
rxFilter.Gain = sum(s.Numerator);
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter(rxIn);
rxFrame = filtOut(rxFilter.FilterSpanInSymbols+1:end);
```

Recover UPs. Display the number of frames lost and the UP cyclic redundancy check (CRC) status.

```
[bits,FramesLost,pktCRCStat] = dvbs2BitRecover(rxFrame,10^(-EsNodB/10));
disp(FramesLost)
```

```
     0
```

```
disp(pktCRCStat)
```

```
    {20×1 logical}
```

Recover Data Bits from Generic Stream DVB-S2 Transmission with Early Termination Enabled

Recover user bits in a multi-input generic stream (GS) Digital Video Broadcasting Satellite Second Generation (DVB-S2) transmission with variable modulation and coding scheme.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
```

```
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
nFrames = 1;
```

Create a DVB-S2 System object with variable coding and modulation configuration for a multi-input GS. Specify the modulation scheme and forward error correction (FEC) rate (MODCOD) and the data field length (DFL).

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.StreamFormat = "GS";
s2WaveGen.NumInputStreams = 3;
s2WaveGen.MODCOD = [10 15 6];    % QPSK 8/9, 8PSK 5/6, and QPSK 2/3
s2WaveGen.DFL = [44500 51387 42960];
```

Create a bit vector of input information bits for each input stream.

```
data =  cell(s2WaveGen.NumInputStreams,1);
for i = 1:s2WaveGen.NumInputStreams
   data{i} = randi([0 1],s2WaveGen.DFL(i)*nFrames,1);
end
```

Generate the DVB-S2 time-domain waveform with the input information bits. Flush the transmit filter to handle the filter delay and recover the complete frame.

```
txWaveform = [s2WaveGen(data); flushFilter(s2WaveGen)];
```

Add additive white Gaussian noise (AWGN) to the generated waveform. Specify the samples per symbol for the baseband filter.

```
sps = s2WaveGen.SamplesPerSymbol;
EsNodB = 10;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn(txWaveform,snrdB,'measured');
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
      'RolloffFactor',s2WaveGen.RolloffFactor, ...
       'InputSamplesPerSymbol',sps,...
       'DecimationFactor',sps);
s = coeffs(rxFilter);
rxFilter.Gain = sum(s.Numerator);
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter(rxIn);
rxFrame = filtOut(rxFilter.FilterSpanInSymbols+1:end);
```

Recover user bits. Enable early termination of the low-density parity-codes (LDPC) decoder.

```
[bits,FramesLost] = dvbs2BitRecover(rxFrame,10^(-EsNodB/10),1);
```

Display the number of frames lost and the number of bit errors in each stream.

```
fprintf('Number of frames lost = %d\n',FramesLost)
```

```
Number of frames lost = 0

for i = 1:s2WaveGen.NumInputStreams
        fprintf('Number of bit errors in stream %d = %d\n',i, ...
            sum(data{i}~=bits{i}))
end

Number of bit errors in stream 1 = 0
Number of bit errors in stream 2 = 0
Number of bit errors in stream 3 = 0
```

Recover Data Bits from Transport Stream DVB-S2 Transmission with ISSYI Enabled

Recover user packets (UPs) in a multi-input transport stream (TS) Digital Video Broadcasting Satellite Second Generation (DVB-S2) transmission with constant coding and modulation.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 1;
```

Create a DVB-S2 System object with constant coding and modulation configuration for a multi-input TS. Specify a short forward error correction (FEC) frame format and enable the input stream synchronization (ISSY).

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.NumInputStreams = 3;
s2WaveGen.FECFrame = "short";
s2WaveGen.MODCOD = 10;       % QPSK 8/9
s2WaveGen.DFL    = 13920;
s2WaveGen.ISSYI = true;
```

Create a bit vector of information bits of concatenated TS UPs.

```
syncBits = [0 1 0 0 0 1 1 1]';      % Sync byte for TS packet is 47 Hex
pktLen = 1496;                      % UP length without sync bits is 1496
data =  cell(1,s2WaveGen.NumInputStreams);
for i = 1:s2WaveGen.NumInputStreams
   numPkts = s2WaveGen.MinNumPackets(i)*numFrames;
   txRawPkts = randi([0 1],pktLen,numPkts);
   ISSY = randi([0 1],16,numPkts);   % ISCRFormat is 'short' by default
                                     % 'short' implies the default length of ISSY as 2 bytes
   txPkts = [repmat(syncBits,1,numPkts); txRawPkts; ISSY]; % ISSY is appended at the end of UP
   data{i} = txPkts(:);
end
```

Generate the DVB-S2 time-domain waveform using the input information bits. Flush the transmit filter to handle the filter delay and recover the complete frame.

```
txWaveform = [s2WaveGen(data); flushFilter(s2WaveGen)];
```

Add additive white Gaussian noise (AWGN) to the generated waveform. Specify the samples per symbol for the baseband filter.

```
sps = s2WaveGen.SamplesPerSymbol;
EsNodB = 12;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn (txWaveform,snrdB,'measured');
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
      'RolloffFactor',s2WaveGen.RolloffFactor, ...
       'InputSamplesPerSymbol',sps,...
       'DecimationFactor', sps);
s = coeffs(rxFilter);
rxFilter.Gain = sum(s.Numerator);
```

Apply matched filtering and remove filter delay.

```
filtOut = rxFilter(rxIn);
rxFrame = filtOut(rxFilter.FilterSpanInSymbols+1:end);
```

Recover UPs. Display the number of frames lost and the number of bit errors in each stream.

```
[bits,FramesLost,pktCRCStat] = dvbs2BitRecover(rxFrame,10^(-EsNodB/10));
fprintf('Number of frames lost = %d\n',FramesLost)
```

```
Number of frames lost = 0
```

```
for i = 1:s2WaveGen.NumInputStreams
    fprintf('Number of bit errors in stream %d = %d\n',i, ...
    numel(pktCRCStat{i})-sum(pktCRCStat{i}))
end
```

```
Number of bit errors in stream 1 = 0
Number of bit errors in stream 2 = 0
Number of bit errors in stream 3 = 0
```

Input Arguments

RXFRAME — Received IQ symbols from PL frames of DVB-S2 transmission

column vector

Received IQ symbols from PL frames of a DVB-S2 single-input or multi-input transmission, specified as a column vector. `RXFRAME` can contain one or multiple PL frames.

The length of `RXFRAME` depends on the value of the properties `FECFrame`, `MODCOD`, and `HasPilots` of the `dvbs2WaveformGenerator` System object™.

Data Types: `double`
Complex Number Support: Yes

NVAR — Noise variance estimate

nonnegative scalar

Noise variance estimate that the function adds to the input IQ symbols, specified as a nonnegative scalar. NVAR is used as a scaling factor to calculate the soft bits from the IQ symbols.

When you specify NVAR as 0, the function uses a value of 1e-5, which corresponds to a signal-to-noise ratio (SNR) of 50 dB.

Data Types: double

EARLYTERM — Flag for early termination of LDPC decoder

0 or false (default) | 1 or true

Flag for early termination of the LDPC decoder when all parity-checks are satisfied, specified as a set logical 1 (true) or 0 (false). When set to 1 (true), the LDPC decoder is terminated when all parity checks are satisfied.

When you set this value to 0 (false), the maximum decoding iteration limit is 50.

Data Types: logical

Output Arguments

BITS — Recovered data bits

cell array of column vectors

Recovered data bits, returned as a cell array of column vectors. Each element of the cell array is of data type int8. This output can be either UPs or generic data stream, depending of the StreamFormat property of the dvbs2WaveformGenerator System object.

For a multi-input stream transmission, each element of the cell array corresponds to an individual input stream.

Data Types: cell

NUMFRAMESLOST — Number of lost baseband frames

nonnegative integer

Number of lost baseband frames, returned as a nonnegative integer. If the baseband header CRC fails, the frame is considered lost.

Data Types: double

PKTCRCSTATUS — UP CRC status

cell array of column vectors

UP CRC status, returned as a cell array of column vectors. Each element of the cell array is of data type logical. For a multi-input stream transmission, each element of the cell array corresponds to an individual input stream.

Dependencies

PKTCRCSTATUS applies for only the input streams where the value of the UPL property of dvbs2WaveformGenerator System object is nonzero.

Data Types: cell

References

[1] ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`dvbs2WaveformGenerator`

Introduced in R2021a

p618PropagationLosses

Calculate Earth-space propagation losses, cross-polarization discrimination, and sky noise temperature

Syntax

```
[pl,xpd,tsky] = p618PropagationLosses(p618cfg)
[pl,xpd,tsky] = p618PropagationLosses(p618cfg,Name,Value)
```

Description

`[pl,xpd,tsky] = p618PropagationLosses(p618cfg)` returns Earth-space propagation losses `pl`, cross-polarization discrimination `xpd`, and sky noise temperature `tsky`, as defined in the ITU-R P.618 recommendation [1]. `p618cfg` specifies the P.618 configuration parameters.

This function requires MAT-files with digital maps from International Telecommunication Union (ITU) documents. If they are not available on the path, download and uncompress the data files from https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz to a location on the MATLAB path.

`[pl,xpd,tsky] = p618PropagationLosses(p618cfg,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Calculate Propagation Losses, Cross-Polarization Discrimination, and Sky Noise Temperature

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and unzip the MAT-files.

```
if ~exist('ITURDigitalMaps.tar.gz', 'file')
    url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
    websave('ITURDigitalMaps.tar.gz',url);
    untar('ITURDigitalMaps.tar.gz');
end
```

Create a default P.618 configuration object.

```
cfg = p618Config;
```

Specify the time percentage of excess for the rain attenuation per annum as 0.01 and the time percentage of excess for the total attenuation per annum as 0.001.

```
cfg.RainAnnualExceedance = 0.01;
cfg.TotalAnnualExceedance = 0.001;
```

Calculate the propagation losses, cross-polarization discrimination, and sky noise temperature.

```
[pl,xpd,tsky] = p618PropagationLosses(cfg)
```

```
pl = struct with fields:
    Ag: 0.2269
    Ac: 0.4552
    Ar: 6.7981
    As: 0.2633
    At: 15.6091


xpd = 32.8876

tsky = 267.4689
```

Calculate Earth-space Propagation Losses Using Name-Value Pair Arguments

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and untar the MAT-files.

```
if ~exist('ITURDigitalMaps.tar.gz','file')
    url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
    websave('ITURDigitalMaps.tar.gz',url);
    untar('ITURDigitalMaps.tar.gz');
end
```

Create a P.618 configuration object with a signal frequency of 20 GHz.

```
cfg = p618Config('Frequency',20e9);
```

Specify the surface water vapor density as $2.8\frac{g}{m^3}$, the total columnar content of the cloud liquid water as 1.4 $\frac{\mathrm{kg}}{m^2}$, and the median value of the wet surface refractivity as 1.2. Set the earth station height as 0.5 km. Calculate the Earth-space propagation losses.

```
pl =  p618PropagationLosses(cfg,'StationHeight',0.5,...
                            'WaterVaporDensity',2.8,...
                            'TotalColumnarContent',1.4,...
                            'WetSurfaceRefractivity',1.2)
```

```
pl = struct with fields:
    Ag: 0.8649
    Ac: 1.0987
    Ar: 0.8907
    As: 0.1372
    At: 2.8590
```

Calculate Propagation Losses in Light Rainfall

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and unzip the MAT-files.

```
if ~exist('ITURDigitalMaps.tar.gz','file')
    url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
```

```
    websave('ITURDigitalMaps.tar.gz',url);
    untar('ITURDigitalMaps.tar.gz');
end
```

Create a P.618 configuration object that occupies a signal frequency of 20 GHz.

```
cfg = p618Config('Frequency',20e9);
```

Calculate the propagation losses in a light rainfall of 1 mm/hr with an earth station height of 0.75 km.

```
pl =  p618PropagationLosses(cfg,'RainRate',1,'StationHeight',0.75)
```

```
pl = struct with fields:
    Ag: 0.7996
    Ac: 0.8793
    Ar: 0.0177
    As: 0.3187
    At: 1.7514
```

Input Arguments

`p618cfg` — P.618 configuration

p618Config object

P.618 configuration required for the calculation of the propagation losses, cross-polarization discrimination, and sky noise temperature, specified as a `p618Config` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'StationHeight',1.5` specifies the earth station height as 1.5 km.

`StationHeight` — Height of earth station

nonnegative scalar

Height of the earth station above the mean sea level in km, specified as the comma-separated pair consisting of '`StationHeight`' and a nonnegative scalar. The maximum supported value is 100. If the local data is not available as an input, the function uses the digital maps provided in ITU-R P.1511 section 1, Annex 1 [3] to obtain the station height value.

Data Types: `double` | `single`

`Temperature` — Temperature of earth surface

nonnegative scalar

Temperature of the earth surface in kelvin, specified as the comma-separated pair consisting of '`Temperature`' and a nonnegative scalar. If the local data is not available as an input, the function uses the map of the mean annual surface temperature provided in ITU-R P.1510 section 1, Annex 1 [4] to obtain the temperature value.

Data Types: `double` | `single`

`Pressure` — Dry air pressure at earth surface
nonnegative scalar

Dry air pressure at the earth surface in hPa, specified as the comma-separated pair consisting of '`Pressure`' and a nonnegative scalar. If the local data is not available as an input, the function uses the mean annual global reference atmosphere provided in ITU-R P.835 section 1.1, Annex 1 [5] to obtain the air pressure value.

Data Types: `double` | `single`

`WaterVaporDensity` — Surface water vapor density
nonnegative scalar

Surface water vapor density in g/m^3, specified as the comma-separated pair consisting of '`WaterVaporDensity`' and a nonnegative scalar. If the local data is not available as an input, the function uses the digital maps provided in ITU-R P.836 section 1, Annex 1 [6] to estimate the value of the water vapor density.

Data Types: `double` | `single`

`IntegratedWaterVaporContent` — Integrated water vapor content
positive scalar

Integrated water vapor content exceeded for the percentage of GasAnnualExceedance in an average year, specified as the comma-separated pair consisting of '`IntegratedWaterVaporContent`' and a positive scalar. Units are in kg/m^2 or mm. If the local data is not available as an input, the function uses the digital maps provided in ITU-R P.836 section 1, Annex 2 [6] to obtain the value of the integrated water vapor content.

Data Types: `double` | `single`

`TotalColumnarContent` — Total columnar content of cloud liquid water
nonnegative scalar

Total columnar content of the cloud liquid water exceeded for the percentage of CloudAnnualExceedance in an average year, specified as the comma-separated pair consisting of '`TotalColumnarContent`' and a nonnegative scalar. Units are in kg/m^2 or mm. If the local data is not available as an input, the function uses the digital maps provided in ITU-R P.840 section 3.1, Annex 1 [7] to obtain the value of the total columnar content.

Data Types: `double` | `single`

`RainRate` — Point rainfall rate
nonnegative scalar

Point rainfall rate at the location for 0.01% of an average year, specified as the comma-separated pair consisting of '`RainRate`' and a nonnegative scalar. Units are in mm/hr. If the local data is not available as an input, the function uses the digital maps provided in ITU-R P.837, Annex 1 [8] to obtain the value of the point rainfall rate.

Data Types: `double` | `single`

`WetSurfaceRefractivity` — Median value of wet term of surface refractivity
nonnegative scalar

Median value of the wet term of the surface refractivity, specified as the comma-separated pair consisting of '`WetSurfaceRefractivity`' and a nonnegative scalar. If the local data is not available

as an input, the function uses the digital maps provided in ITU-R P.453 section 2.2, Annex 1 [9] to obtain the value of the wet surface refractivity.

Data Types: `double` | `single`

`MeanRadiatingTemperature` — Atmospheric mean radiating temperature
nonnegative scalar

Atmospheric mean radiating temperature in kelvin, specified as the comma-separated pair consisting of '`MeanRadiatingTemperature`' and a nonnegative scalar. If the local data is not available as an input, the function uses an atmospheric mean radiating temperature of 275 K in the computation.

Data Types: `double` | `single`

Output Arguments

`pl` — Earth-space propagation losses information
structure

Earth-space propagation losses information, returned as a structure containing these fields.

Fields	Description
At	Total atmospheric attenuation (in dB)
Ag	Gaseous attenuation (in dB)
Ac	Cloud and fog attenuation (in dB)
Ar	Rain attenuation (in dB)
As	Attenuation due to tropospheric scintillation (in dB)

`xpd` — Cross-polarization discrimination
scalar

Cross-polarization discrimination in (dB) not exceeded for the percentage of the RainAnnualExceedance, returned as a scalar.

`tsky` — Sky noise temperature
nonnegative scalar

Sky noise temperature (in kelvin) at the ground station antenna, returned as a nonnegative scalar.

References

[1] International Telecommunication Union, ITU-R Recommendation P.618 (12/2017).

[2] International Telecommunication Union, ITU-R Recommendation P.676 (08/2019).

[3] International Telecommunication Union, ITU-R Recommendation P.1511 (08/2019).

[4] International Telecommunication Union, ITU-R Recommendation P.1510 (06/2017).

[5] International Telecommunication Union, ITU-R Recommendation P.835 (12/2017).

[6] International Telecommunication Union, ITU-R Recommendation P.836 (12/2017).

[7] International Telecommunication Union, ITU-R Recommendation P.840 (08/2019).

[8] International Telecommunication Union, ITU-R Recommendation P.837 (06/2017).

[9] International Telecommunication Union, ITU-R Recommendation P.453 (08/2019).

[10] International Telecommunication Union, ITU-R Recommendation P.839 (09/2013).

[11] International Telecommunication Union, ITU-R Recommendation P.838 (03/2005).

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Supports only MEX code generation.

See Also

Objects

`p618Config` | `p618SiteDiversityConfig`

Functions

`p618SiteDiversityOutage`

Introduced in R2021a

p618SiteDiversityOutage

Calculate outage probability due to rain attenuation with site diversity

Syntax

```
Outage = p618SiteDiversityOutage(cfgsd)
Outage = p618SiteDiversityOutage(cfgsd,Name,Value)
```

Description

`Outage = p618SiteDiversityOutage(cfgsd)` returns the outage probability due to rain attenuation with site diversity. The function calculates this value as per the ITU-R P.618 recommendation [1].

This function requires MAT-files with digital maps from International Telecommunication Union (ITU) documents. If they are not available on the path, download and uncompress the data files from https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz to a location on the MATLAB path.

`Outage = p618SiteDiversityOutage(cfgsd,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Calculate Outage Probability due to Rain Attenuation with Site Diversity

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and untar the MAT-files.

```
if ~exist('ITURDigitalMaps.tar.gz','file')
    url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
    websave('ITURDigitalMaps.tar.gz',url);
    untar('ITURDigitalMaps.tar.gz');
end
```

Create a P.618 site diversity configuration object with a signal frequency of 25 GHz.

```
cfgsd = p618SiteDiversityConfig;
cfgsd.Frequency = 25e9;
```

Specify the polarization tilt angles for two sites as [-90 90] degrees, separation between the two sites as 50 km, and attenuation threshold on the two links as [9 9] dB.

```
cfgsd.PolarizationTiltAngle = [-90 90];
cfgsd.SiteDistance = 50;
cfgsd.AttenuationThreshold = [9 9];
```

Calculate the outage probability due to rain attenuation with site diversity.

```
outage = p618SiteDiversityOutage(cfgsd)
```

```
outage = 0.0338
```

Calculate Outage Probability with Site Diversity Using Name-Value Pair Arguments

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute these commands to download and untar the MAT-files.

```
if ~exist('ITURDigitalMaps.tar.gz','file')
    url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
    websave('ITURDigitalMaps.tar.gz',url);
    untar('ITURDigitalMaps.tar.gz');
end
```

Create a default P.618 site diversity configuration object. Change the signal frequency to 25 GHz.

```
cfgsd = p618SiteDiversityConfig;
cfgsd.Frequency = 25e9;
```

Specify the separation between two sites as 50 km and the attenuation threshold on the two links as [9 9] dB.

```
cfgsd.SiteDistance = 50;
cfgsd.AttenuationThreshold = [9 9];
```

Calculate the outage probability for the specified site diversity configuration.

```
outage = p618SiteDiversityOutage(cfgsd,'RainAnnualExceedances',[0.01 0.05 0.2],...
                                 'RainProbability1',0.3,...
                                 'RainProbability2',0.5)
```

```
outage = 0.0339
```

Input Arguments

`cfgsd` — P.618 site diversity configuration

p618SiteDiversityConfig object

P.618 site diversity configuration required for the calculation of the outage probability due to rain attenuation, specified as a `p618SiteDiversityConfig` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'RainAnnualExceedances',[0.01 0.02 0.03 0.05]` specifies the average annual time percentage of excess for the rain attenuation.

`RainAnnualExceedances` — Average annual time percentage of excess for rain attenuation

nonnegative vector

Average annual time percentage of excess for the rain attenuation, specified as the comma-separated pair consisting of `'RainAnnualExceedances'` and a nonnegative vector. The values in this vector must be less than the probability of rain at the two sites.

If the local data is not available as an input, the function uses `[0.01 0.02 0.03 0.05 0.1 0.2 0.3 0.5 1 2 3 5]` as the default vector.

Data Types: `double` | `single`

`RainAttenuations1` — Rain attenuations at site 1
nonnegative vector

Rain attenuations (in dB) at site 1, specified as the comma-separated pair consisting of `'RainAttenuations1'` and a nonnegative vector. This value specifies the rain attenuation exceeded for the percentages given in the `RainAnnualExceedances` name-value pair argument. The dimension of this value must match that of the `RainAnnualExceedances`.

If the local data is not available as an input, the function uses the method as defined in section 2.2.1.1 of the ITU-R P.618 [1] recommendation to calculate the rain attenuations for site 1.

Note If you do not specify `RainAttenuations1`, then `RainAnnualExceedances` must be in the range from 0.01% to 5%.

Data Types: `double` | `single`

`RainAttenuations2` — Rain attenuations at site 2
nonnegative vector

Rain attenuations (in dB) at site 2, specified as the comma-separated pair consisting of `'RainAttenuations2'` and a nonnegative vector. This value specifies the rain attenuation exceeded for the percentages given in the `RainAnnualExceedances` name-value pair argument. The dimension of this value must match that of the `RainAnnualExceedances`.

If the local data is not available as an input, the function uses the method as defined in section 2.2.1.1 of the ITU-R P.618 recommendation to calculate the rain attenuations for site 2.

Note If you do not specify `RainAttenuations2`, then `RainAnnualExceedances` must be in the range from 0.01% to 5%.

Data Types: `double` | `single`

`RainProbabilty1` — Probability of rain for site 1
nonnegative scalar

Probability of (in %) rain for site 1, specified as the comma-separated pair consisting of `'RainProbabilty1'` and a nonnegative scalar.

If the local measured rainfall rate data is not available as an input, the function uses the digital maps as defined in ITU-R P.837 Annex 1 [2] to calculate the rain probability for the sites.

Data Types: `double` | `single`

`RainProbabilty2` — Probability of rain for site 2
nonnegative scalar

Probability of (in %) rain for site 2, specified as the comma-separated pair consisting of `'RainProbabilty2'` and a nonnegative scalar.

If the local measured rainfall rate data is not available as an input, the function uses the digital maps as defined in ITU-R P.837 Annex 1 [2] to calculate the rain probability for the sites.

Data Types: `double` | `single`

Output Arguments

`Outage` — Outage probability due to rain attenuation with site diversity
nonnegative scalar

Outage probability due to rain attenuation with site diversity, returned as a nonnegative scalar. This argument predicts the joint probability ($P_r(A_1 \geq a_1, A_2 \geq a_2)$), where the attenuation on the path of the site 1 must exceed a_1 and the attenuation on the path of the site 2 must exceed a_2.

References

[1] International Telecommunication Union, ITU-R Recommendation P.618 (12/2017).

[2] International Telecommunication Union, ITU-R Recommendation P.837 (06/2017).

[3] International Telecommunication Union, ITU-R Recommendation P.1511 (08/2019).

[4] International Telecommunication Union, ITU-R Recommendation P.1510 (06/2017).

[5] International Telecommunication Union, ITU-R Recommendation P.836 (12/2017).

[6] International Telecommunication Union, ITU-R Recommendation P.840 (08/2019).

[7] International Telecommunication Union, ITU-R Recommendation P.453 (08/2019).

[8] International Telecommunication Union, ITU-R Recommendation P.839 (09/2013).

[9] International Telecommunication Union, ITU-R Recommendation P.838 (03/2005).

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Supports only MEX code generation.

See Also

Objects
`p618Config` | `p618SiteDiversityConfig`

Functions
p618PropagationLosses

Introduced in R2021a

ccsdsTCWaveform

Generate CCSDS TC waveform

Syntax

```
waveform = ccsdsTCWaveform(bits,cfgFormat)
[waveform,encodedBits] = ccsdsTCWaveform(bits,cfgFormat)
```

Description

`waveform = ccsdsTCWaveform(bits,cfgFormat)` generates a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) time-domain waveform, `waveform`, for the corresponding input bits, `bits`, and the given format configuration, `cfgFormat`.

`[waveform,encodedBits] = ccsdsTCWaveform(bits,cfgFormat)` also returns the bits obtained after TC synchronization and channel coding sublayer operations.

Examples

Create CCSDS TC Waveform for Multiple CLTUs

Create a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) time-domain waveform for multiple communications link transmission units (CLTUs).

Create a default CCSDS TC configuration object.

```
cfg = ccsdsTCConfig;
disp(cfg)
```

```
  ccsdsTCConfig with properties:

            DataFormat: "CLTU"
         ChannelCoding: "BCH"
         HasRandomizer: 1
            Modulation: "PCM/PSK/PM"
             PCMFormat: "NRZ-L"
       ModulationIndex: 0.4000
    SubcarrierFrequency: 16000
            SymbolRate: 4000
       SamplesPerSymbol: 10

   Read-only properties:
     SubcarrierWaveform: "sine"
```

Specify the number of CLTUs and the transfer frame length.

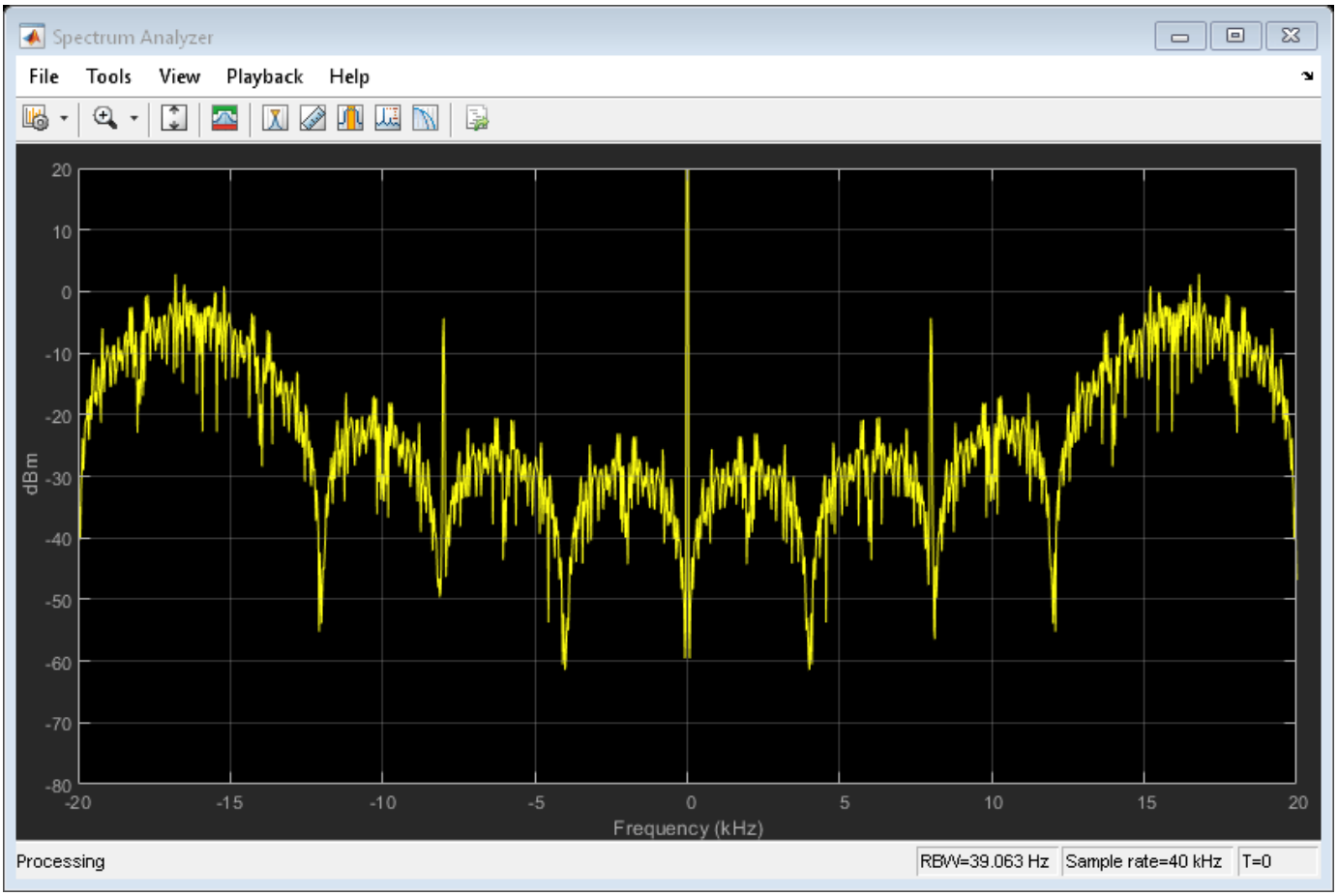
```
numCLTUs = 10;
transferFramesLength = 8; % Number of octets in each transfer frame
```

Generate the CCSDS TC time-domain waveform for the transfer frames.

```
c = cell(1,numCLTUs); % Cell array to store the generated waveform for all CLTUs
for k=1:numCLTUs
    bits = randi([0 1],8*transferFramesLength,1); % Bits in the TC transfer frame
    waveform = ccsdsTCWaveform(bits,cfg);
    c{1,k} = waveform; % Waveform for each CLTU
end
```

Create a `dsp.SpectrumAnalyzer` System object to display the frequency spectrum of the generated CCSDS TC time-domain waveform from the last CLTU.

```
scope = dsp.SpectrumAnalyzer;
scope.SampleRate = cfg.SamplesPerSymbol*cfg.SymbolRate;
scope(waveform)  % Last CLTU spectrum display
```



Create CCSDS TC Waveform for Acquisition Sequence

Create a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) time-domain waveform for a acquisition sequence with 20 octets.

Create a CCSDS TC configuration object, and then specify the object properties. Display the object properties.

```
cfg = ccsdsTCConfig;
cfg.DataFormat = "acquisition sequence";
cfg.Modulation = "PCM/PM/biphase-L";
cfg.ModulationIndex = 1.2;
disp(cfg)
```

```
  ccsdsTCConfig with properties:

         DataFormat: "acquisition sequence"
         Modulation: "PCM/PM/biphase-L"
    ModulationIndex: 1.2000
   SamplesPerSymbol: 10
```

Generate the CCSDS TC waveform.

```
bits = repmat([0;1],8*10,1); % Alternating 1s and 0s with 0s as a starting sequence bit
waveform = ccsdsTCWaveform(bits,cfg);
```

Input Arguments

`bits` — Information bits

binary-valued column vector

Information bits, specified as a binary-valued column vector.

- When you set the DataFormat property of the `ccsdsTCConfig` object to `"CLTU"`, the length of this vector must be an integer multiple of 8.
- When you set the `DataFormat` property of the `ccsdsTCConfig` object to `"acquisition sequence"` or `"idle sequence"`, this vector must be a sequence of alternating 1s and 0s, starting with either 1 or 0.

Data Types: `double` | `int8` | `logical`

`cfgFormat` — Format configuration object

`ccsdsTCConfig` object

Format configuration object, specified as `ccsdsTCConfig` object. The properties of this object define the parameters required for CCSDS TC waveform generation.

Output Arguments

`waveform` — Generated time-domain CCSDS TC waveform

column vector

Generated time-domain CCSDS TC waveform, returned as a column vector. The `waveform` output is generated in the form of complex in-phase quadrature (IQ) samples.

Data Types: `double`

`encodedBits` — Output bits obtained after TC synchronization and channel coding sublayer operations

column vector

Output bits obtained after TC synchronization and channel coding sublayer operations, returned as a column vector.

Data Types: `double`

References

[1] CCSDS 231.0-B-3. Blue Book. Issue 3. "TC Synchronization and Channel Coding." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, September 2017.

[2] CCSDS 401.0-B-29. Blue Book. Issue 29. "Radio Frequency and Modulation Systems - Part 1". *Earth Stations and Spacecraft*. Washington, D.C.: CCSDS, September 2019.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`ccsdsTCIdealReceiver`

Objects

`ccsdsTCConfig`

Introduced in R2021a

ccsdsTCIdealReceiver

Ideal receiver for CCSDS TC waveform

Syntax

```
bits = ccsdsTCIdealReceiver(waveform,cfg)
bits = ccsdsTCIdealReceiver(waveform,cfg,Name,Value)
```

Description

`bits = ccsdsTCIdealReceiver(waveform,cfg)` recovers transfer frames from a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) `waveform`, generated using the `ccsdsTCWaveform` function. Output `bits` is the recovered bits for the given format configuration `cfg`.

`bits = ccsdsTCIdealReceiver(waveform,cfg,Name,Value)` specifies options using one or more name-value pairs. For example, `'NoiseVariance',1e-11` specifies the noise variance of additive white Gaussian noise (AWGN) on the received waveform as 1e-11.

Examples

Recover Transfer Frame from CCSDS TC Waveform

Recover the transfer frame from the Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) waveform.

Create a CCSDS TC object and specify the object properties.

```
cfg = ccsdsTCConfig;
cfg.HasRandomizer = 1;
cfg.SamplesPerSymbol = 12;
disp(cfg)
```

```
  ccsdsTCConfig with properties:

            DataFormat: "CLTU"
         ChannelCoding: "BCH"
         HasRandomizer: 1
            Modulation: "PCM/PSK/PM"
             PCMFormat: "NRZ-L"
       ModulationIndex: 0.4000
    SubcarrierFrequency: 16000
            SymbolRate: 4000
       SamplesPerSymbol: 12

   Read-only properties:
     SubcarrierWaveform: "sine"
```

Specify the transfer frame length and generate the CCSDS TC waveform for the transfer frame.

```
transferFrameLength = 12; % Number of octets in each transfer frame
data = randi([0 1],8*transferFrameLength,1); % bits in the transfer frame
waveform = ccsdsTCWaveform(data,cfg);
```

Recover the transfer frame from the CCSDS TC waveform

```
decodedBits = ccsdsTCIdealReceiver(waveform,cfg,'DecodingMode',"error detecting");
```

Check if the transfer frame is recovered successfully.

```
rxBits = decodedBits{1};
bits = rxBits((1:8*transferFrameLength)');
isequal(bits,data)
```

```
ans = logical
   1
```

Input Arguments

`waveform` — Received time-domain signal

column vector

Received time-domain signal, consisting of complex in-phase quadrature (IQ) samples, specified as a column vector. The `waveform` input is a CCSDS TC waveform.

A CCSDS TC waveform can contain one or more communications link transmission units (CLTUs). Each CLTU can contain one or more transfer frames.

Data Types: `single` | `double`
Complex Number Support: Yes

`cfg` — Format configuration object

`ccsdsTCConfig` object

Format configuration object, specified as `ccsdsTCConfig` object. The properties of this object determine the parameters required for CCSDS TC waveform generation and reception.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `ccsdsIdealReceiver(waveform,cfg,'NoiseVariance',1e-11)` specifies the noise variance of AWGN on the received waveform as 1e-11.

`NoiseVariance` — Noise variance of AWGN

`1e-10` (default) | positive scalar

Noise variance of AWGN that is added to the input IQ symbols of the waveform, specified as a positive scalar.

Dependencies

To enable this name-value pair, set the `ChannelCoding` property of the `cfg` input to `"LDPC"`.

Data Types: `double`

`DecodingMode` — Decoding mode
`"error correcting"` (default) | `"error detecting"`

Decoding mode to decode the Bose Chaudhuri Hocquenghem (BCH) encoded codewords, specified as `"error correcting"` or `"error detecting"`.

`'DecodingMode'` defines the allowed number of errors in the start sequence of the CLTU. In error detecting mode, the allowed number of errors in the start sequence is zero. In error correcting mode, the allowed number of errors in the start sequence is one.

Dependencies

To enable this name-value pair, set the `ChannelCoding` property of the `cfg` input to `"BCH"`.

Data Types: `char` | `string`

`DetectionThreshold` — Threshold to detect start sequence
`0.7` (default) | scalar in the range [0.5, 1]

Threshold to detect the start sequence, by calculating the normalized correlation metric with the known start sequence, specified as a scalar in the range [0.5, 1]. When the computed normalized correlation metric is greater than or equal to `'DetectionThreshold'`, the start sequence of the CLTU is detected.

Dependencies

To enable this name-value pair, set the `ChannelCoding` property of the `cfg` input to `"LDPC"`.

Data Types: `double`

Output Arguments

`bits` — Recovered transfer frames
cell array of column vectors

Recovered transfer frames, returned as a cell array of column vectors. Each element of the cell array is of data type `int8`.

Bits in the cell array of one or more column vectors, corresponds to the number of CLTUs present in the `waveform` input. Recovered transfer frames of CLTUs can contain fill bits. The fill bits removal procedure is not performed in the TC synchronization and channel coding sublayer.

Data Types: `int8` | `cell`

References

[1] CCSDS 231.0-B-3. Blue Book. Issue 3. "TC Synchronization and Channel Coding." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, September 2017.

[2] CCSDS 401.0-B-29. Blue Book. Issue 29. "Radio Frequency and Modulation Systems - Part 1". *Earth Stations and Spacecraft*. Washington, D.C.: CCSDS, September 2019.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`ccsdsTCWaveform`

Objects

`ccsdsTCConfig`

Introduced in R2021a

info

Characteristic information about object

Syntax

```
s = info(obj)
```

Description

`s = info(obj)` returns a structure containing the characteristic information of the specified input object `obj`.

Examples

Get DVB-S2 Waveform Generator Information and Check Transmit Filter Delay

Get information from a `dvbs2WaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip', 'file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 1;
```

Create a Digital Video Broadcasting standard (DVB-S2) System object, and then specify its properties.

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.NumInputStreams = 2;
s2WaveGen.MODCOD = [21 16];
s2WaveGen.DFL   = 47008;
s2WaveGen.ISSYI = true;
s2WaveGen.SamplesPerSymbol = 2;
disp(s2WaveGen)
```

```
  dvbs2WaveformGenerator with properties:

          StreamFormat: "TS"
       NumInputStreams: 2
               FECFrame: "normal"
                 MODCOD: [21 16]
```

```
                    DFL: 47008
          ScalingMethod: "outer radius as 1"
              HasPilots: 0
          RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 2
                  ISSYI: true
             ISCRFormat: "short"

  Show all properties
```

Get the characteristic information about the DVB-S2 waveform generator.

```
info(s2WaveGen)
```

```
ans = struct with fields:
      ModulationScheme: {'16APSK'  '8PSK'}
    LDPCCodeIdentifier: {'5/6'  '8/9'}
```

Create the bit vector of input information bits, `data`, of concatenated TS user packets.

```
syncBits = [0 1 0 0 0 1 1 1]';       % Sync byte for TS packet is 47 Hex
pktLen = 1496;                       % UP length without sync bits is 1496
data =  cell(1,s2WaveGen.NumInputStreams);
for i = 1:s2WaveGen.NumInputStreams
    numPkts = s2WaveGen.MinNumPackets(i)*numFrames;
    txRawPkts = randi([0 1],pktLen,numPkts);
    ISSY = randi([0 1],16,numPkts);   % ISCRFormat is 'short' by default
                                      % 'short' implies the default length of ISSY as 2 bytes
    txPkts = [repmat(syncBits,1,numPkts);txRawPkts;ISSY];    % ISSY is appended at the end of UP
    data{i} = txPkts(:);
end
```

Generate a DVB-S2 time-domain waveform using the information bits.

```
txWaveform = [s2WaveGen(data)];
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(s2WaveGen)
```

```
ans = 20×1 complex

   0.0153 + 0.4565i
   0.2483 + 0.5535i
   0.3527 + 0.3972i
   0.3541 - 0.0855i
   0.3505 - 0.4071i
   0.4182 - 0.1962i
   0.5068 + 0.0636i
   0.4856 - 0.1532i
   0.3523 - 0.4153i
   0.1597 - 0.2263i
      ⋮
```

Get DVB-S2X Waveform Generator Information and Check Transmit Filter Delay

Get information from a `dvbs2xWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 2;
```

Create a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) System object and specify its properties. Use time slicing technique and variable coding and modulation configuration mode.

```
s2xWaveGen = dvbs2xWaveformGenerator();
s2xWaveGen.HasTimeSlicing = true;
s2xWaveGen.NumInputStreams = 2;
s2xWaveGen.PLSDecimalCode = [135 145];   % QPSK 9/20 and 8PSK 25/36
s2xWaveGen.DFL = [18048 44656];
s2xWaveGen.PLScramblingIndex = [0 1];
disp(s2xWaveGen)
```

```
  dvbs2xWaveformGenerator with properties:

          StreamFormat: "TS"
        HasTimeSlicing: true
       NumInputStreams: 2
         PLSDecimalCode: [135 145]
                    DFL: [18048 44656]
     PLScramblingIndex: [0 1]
          RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 4
                  ISSYI: false

  Show all properties
```

Get the characteristic information about the DVB-S2X waveform generator.

```
info(s2xWaveGen)
```

```
ans = struct with fields:
              FECFrame: {'normal'  'normal'}
      ModulationScheme: {'QPSK'  '8PSK'}
    LDPCCodeIdentifier: {'9/20'  '25/36'}
```

Create the bit vector of input information bits, `data`, of concatenated TS user packets for each input stream.

```
syncBits = [0 1 0 0 0 1 1 1]';       % Sync byte for TS packet is 47 Hex
pktLen = 1496;                       % UP length without sync bits is 1496
data =  cell(1, s2xWaveGen.NumInputStreams);
for i = 1:s2xWaveGen.NumInputStreams
    numPkts = s2xWaveGen.MinNumPackets(i)*numFrames;
    txRawPkts = randi([0 1], pktLen, numPkts);
    txPkts = [repmat(syncBits, 1, numPkts); txRawPkts];
    data{i} = txPkts(:);
end
```

Generate a DVB-S2X time-domain waveform using the information bits.

```
txWaveform = s2xWaveGen(data);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(s2xWaveGen)
```

```
ans = 40×1 complex

  -0.2412 - 0.0143i
  -0.2619 - 0.0861i
  -0.2726 - 0.1337i
  -0.2511 - 0.1597i
  -0.1851 - 0.1680i
  -0.0780 - 0.1602i
   0.0448 - 0.1288i
   0.1598 - 0.0751i
   0.2482 - 0.0049i
   0.3026 + 0.0702i
      ⋮
```

Get CCSDS TM Waveform Generator Information and Check Transmit Filter Delay

Get information from a `ccsdsTMWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

Create a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) System object. Set the waveform type as `synchronization and channel coding` with low-density parity-check (LDPC) channel coding. Display the properties.

```
tmWaveGen = ccsdsTMWaveformGenerator;
tmWaveGen.WaveformSource = "synchronization and channel coding";
tmWaveGen.ChannelCoding = "LDPC";
tmWaveGen.NumBitsInInformationBlock = 1024;
tmWaveGen.Modulation = "QPSK";
tmWaveGen.CodeRate = "1/2";
disp(tmWaveGen)
```

```
  ccsdsTMWaveformGenerator with properties:
```

```
              WaveformSource: "synchronization and channel coding"
               HasRandomizer: true
                      HasASM: true
                   PCMFormat: "NRZ-L"

   Channel coding properties:
               ChannelCoding: "LDPC"
    NumBitsInInformationBlock: 1024
                     CodeRate: "1/2"
                IsLDPCOnSMTF: false

   Digital modulation and filter properties:
                  Modulation: "QPSK"
           PulseShapingFilter: "root raised cosine"
                RolloffFactor: 0.3500
          FilterSpanInSymbols: 10
             SamplesPerSymbol: 10

  Use get to show all properties
```

Specify the number of transfer frames.

```
numTF = 20;
```

Get the characteristic information about the CCSDS TM waveform generator.

```
info(tmWaveGen)
```

```
ans = struct with fields:
         ActualCodeRate: 0.5000
       NumBitsPerSymbol: 2
    SubcarrierFrequency: []
```

Generate the input bits for the CCSDS TM waveform generator, and then generate the waveform.

```
bits = randi([0 1], tmWaveGen.NumInputBits*numTF,1);
waveform = tmWaveGen(bits);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(tmWaveGen)
```

```
ans = 100×1 complex

  -0.0772 - 0.0867i
  -0.0751 - 0.0859i
  -0.0673 - 0.0788i
  -0.0549 - 0.0654i
  -0.0388 - 0.0469i
  -0.0200 - 0.0250i
   0.0002 - 0.0012i
   0.0208 + 0.0227i
   0.0405 + 0.0453i
   0.0587 + 0.0653i
      ⋮
```

Get ETSI Rician Channel Information

Get information from a `etsiRicianChannel` System object by using the `info` object function.

Create a European Telecommunication Standards Institute (ETSI) Rician channel System object, and then specify its properties.

```
chan = etsiRicianChannel;
chan.SampleRate = 2e5;
chan.KFactor = 10;
chan.MaximumDopplerShift = 20;
chan.NumSinusoids = 58;
disp(chan)
```

```
  etsiRicianChannel with properties:

            SampleRate: 200000
               KFactor: 10
    MaximumDopplerShift: 20

  Use get to show all properties
```

Pass data through the channel.

```
txWaveform = randi([0 1],500,1);
rxWaveform = chan(txWaveform);
```

Get the characteristic information about the ETSI Rician channel.

```
info(chan)
```

```
ans = struct with fields:
           ChannelFilterDelay: 0
    ChannelFilterCoefficients: 1
          NumSamplesProcessed: 500
```

Input Arguments

`obj` — Input object

`dvbs2WaveformGenerator` | `dvbs2xWaveformGenerator` | `etsiRicianChannel` | `ccsdsTMWaveformGenerator`

Input object to get information from, specified as a `dvbs2WaveformGenerator`, `dvbs2xWaveformGenerator`, `ccsdsTMWaveformGenerator`, or `etsiRicianChannel` System object.

Output Arguments

`s` — Characteristic information of specified object

structure

Characteristic information of the specified object, returned as a structure. The fields of the structure depends on the `obj` input.

- If `obj` is a `dvbs2WaveformGenerator` System object, the output structure has these fields, consisting of physical layer information about the DVB-S2 waveform generator.

Field	Value	Description
`ModulationScheme`	String scalar (default) or cell array of character vectors	Modulation scheme, returned as a string scalar for single-input stream and a cell array of character vectors of length equal to the `NumInputStreams` property of the `dvbs2WaveformGenerator` object for multi-input streams.
`LDPCCodeIdentifier`	String scalar (default) or cell array of character vectors	LDPC code identifier used in forward error correction (FEC), returned as a string scalar for single-input stream and a cell array of character vectors of length equal to `NumInputStreams` property of the `dvbs2WaveformGenerator` object for multi-input streams.

- If `obj` is a `dvbs2xWaveformGenerator` System object, the output structure has these fields, consisting of physical layer information about the DVB-S2X waveform generator.

Field	Value	Description
`FECFrame`	String scalar (default) or cell array of character vectors	FEC frame format, returned as a string scalar for single-input stream and a cell array of character vectors of length equal to `NumInputStreams` property of `dvbs2xWaveformGenerator` object for multi-input streams.
`ModulationScheme`	String scalar (default) or cell array of character vectors	Modulation scheme, returned as a string scalar for single-input stream and a cell array of character vectors of length equal to `NumInputStreams` property of `dvbs2xWaveformGenerator` object for multi-input streams.

Field	Value	Description
`LDPCCodeIdentifier`	String scalar (default) or cell array of character vectors	LDPC code identifier used in forward error correction (FEC), returned as a string scalar for single-input stream and a cell array of character vectors of length equal to `NumInputStreams` property of `dvbs2xWaveformGenerator` object for multi-input streams.

- If `obj` is an `etsiRicianChannel` System object, the output structure has these fields, consisting of information about the fading channel.

Field	Value	Description
`ChannelFilterDelay`	`0`	Channel filter delay in samples returned as `0` always (due to flat-fading nature of the channel).
`ChannelFilterCoefficients`	`1`	Channel filter coefficient used to convert path gains to channel filter tap gains, returned as `1` always (as `etsiRicianChannel` describes a single path channel).
`NumSamplesProcessed`	positive integer	Number of samples processed by the channel object since the last reset, returned as a positive integer.

- If `obj` is a `ccsdsTMWaveformGenerator` System object, the output structure has these fields, consisting of physical layer information about the CCSDS TM waveform generator.

Field	Value	Description
`ActualCodeRate`	positive scalar in range [0 1]	Numeric value of the code rate of the channel coding scheme, retuned as a positive scalar in the range [0, 1]. This value is used to generate the CCSDS TM waveform.
`NumBitsPerSymbol`	positive integer	Number of bits per modulated symbol, returned as a positive integer.

Field	Value	Description
`SubcarrierFrequency`	positive scalar	Subcarrier frequency, returned as a positive scalar. This field is applicable only when `Modulation` property of `ccsdsTMWaveformGenerator` object is set to `"PCM/PSK/PM"`. For other cases, this value is returned as null.

See Also

Functions
`flushFilter`

Objects
`ccsdsTMWaveformGenerator` | `dvbs2WaveformGenerator` | `dvbs2xWaveformGenerator` | `etsiRicianChannel`

Introduced in R2021a

flushFilter

Flush transmit filter

Syntax

```
out = flushFilter(obj)
```

Description

`out = flushFilter(obj)` passes zeros through the transmit filter in the input waveform generator to flush the residual data samples that remain in the filter state. The function returns the residual data samples.

You must call the input waveform generator System object (not only create the object) prior to using the `flushFilter` object function. The number of zeros passed through the transmit filter depends on the filter delay. This object function is required for the receiver simulations to recover all of the bits in the last physical layer frame.

Examples

Get DVB-S2 Waveform Generator Information and Check Transmit Filter Delay

Get information from a `dvbs2WaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip', 'file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 1;
```

Create a Digital Video Broadcasting standard (DVB-S2) System object, and then specify its properties.

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.NumInputStreams = 2;
s2WaveGen.MODCOD = [21 16];
s2WaveGen.DFL    = 47008;
s2WaveGen.ISSYI = true;
s2WaveGen.SamplesPerSymbol = 2;
disp(s2WaveGen)
```

```
  dvbs2WaveformGenerator with properties:

          StreamFormat: "TS"
       NumInputStreams: 2
               FECFrame: "normal"
                 MODCOD: [21 16]
                    DFL: 47008
         ScalingMethod: "outer radius as 1"
             HasPilots: 0
         RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 2
                  ISSYI: true
             ISCRFormat: "short"

  Show all properties
```

Get the characteristic information about the DVB-S2 waveform generator.

```
info(s2WaveGen)
```

```
ans = struct with fields:
      ModulationScheme: {'16APSK'  '8PSK'}
    LDPCCodeIdentifier: {'5/6'  '8/9'}
```

Create the bit vector of input information bits, `data`, of concatenated TS user packets.

```
syncBits = [0 1 0 0 0 1 1 1]';       % Sync byte for TS packet is 47 Hex
pktLen = 1496;                       % UP length without sync bits is 1496
data =  cell(1,s2WaveGen.NumInputStreams);
for i = 1:s2WaveGen.NumInputStreams
    numPkts = s2WaveGen.MinNumPackets(i)*numFrames;
    txRawPkts = randi([0 1],pktLen,numPkts);
    ISSY = randi([0 1],16,numPkts);   % ISCRFormat is 'short' by default
                                      % 'short' implies the default length of ISSY as 2 bytes
    txPkts = [repmat(syncBits,1,numPkts);txRawPkts;ISSY];    % ISSY is appended at the end of UP
    data{i} = txPkts(:);
end
```

Generate a DVB-S2 time-domain waveform using the information bits.

```
txWaveform = [s2WaveGen(data)];
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(s2WaveGen)
```

```
ans = 20×1 complex

   0.0153 + 0.4565i
   0.2483 + 0.5535i
   0.3527 + 0.3972i
   0.3541 - 0.0855i
   0.3505 - 0.4071i
   0.4182 - 0.1962i
   0.5068 + 0.0636i
   0.4856 - 0.1532i
   0.3523 - 0.4153i
```

```
   0.1597 - 0.2263i
      ⋮
```

Recover Data Bits from Transport Stream DVB-S2 Transmission

Recover user packets (UPs) for multiple physical layer (PL) frames in a single transport stream Digital Video Broadcasting Satellite Second Generation (DVB-S2) transmission.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream. Create a DVB-S2 System object.

```
nFrames = 2;
s2WaveGen = dvbs2WaveformGenerator;
```

Create the bit vector of information bits, `data`, of concatenated TS UPs.

```
syncBits = [0 1 0 0 0 1 1 1]';    % Sync byte for TS packet is 47 Hex
pktLen = 1496;                    % UP length without sync bits is 1496
numPkts = s2WaveGen.MinNumPackets*nFrames;
txRawPkts = randi([0 1],pktLen,numPkts);
txPkts = [repmat(syncBits,1,numPkts); txRawPkts];
data = txPkts(:);
```

Generate the DVB-S2 time-domain waveform using the input information bits. Flush the transmit filter to handle the filter delay and recover the complete last frame.

```
txWaveform = [s2WaveGen(data); flushFilter(s2WaveGen)];
```

Add additive white Gaussian noise (AWGN) to the generated waveform.

```
sps = s2WaveGen.SamplesPerSymbol;
EsNodB = 1;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn(txWaveform,snrdB,'measured');
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
      'RolloffFactor',s2WaveGen.RolloffFactor, ...
       'InputSamplesPerSymbol',sps,...
       'DecimationFactor',sps);
s = coeffs(rxFilter);
rxFilter.Gain = sum(s.Numerator);
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter(rxIn);
rxFrame = filtOut(rxFilter.FilterSpanInSymbols+1:end);
```

Recover UPs. Display the number of frames lost and the UP cyclic redundancy check (CRC) status.

```
[bits,FramesLost,pktCRCStat] = dvbs2BitRecover(rxFrame,10^(-EsNodB/10));
disp(FramesLost)
```

```
     0
```

```
disp(pktCRCStat)
```

```
    {20×1 logical}
```

Get DVB-S2X Waveform Generator Information and Check Transmit Filter Delay

Get information from a `dvbs2xWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 2;
```

Create a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) System object and specify its properties. Use time slicing technique and variable coding and modulation configuration mode.

```
s2xWaveGen = dvbs2xWaveformGenerator();
s2xWaveGen.HasTimeSlicing = true;
s2xWaveGen.NumInputStreams = 2;
s2xWaveGen.PLSDecimalCode = [135 145];   % QPSK 9/20 and 8PSK 25/36
s2xWaveGen.DFL = [18048 44656];
s2xWaveGen.PLScramblingIndex = [0 1];
disp(s2xWaveGen)
```

```
  dvbs2xWaveformGenerator with properties:

          StreamFormat: "TS"
        HasTimeSlicing: true
       NumInputStreams: 2
        PLSDecimalCode: [135 145]
                   DFL: [18048 44656]
```

```
      PLScramblingIndex: [0 1]
          RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 4
                  ISSYI: false

  Show all properties
```

Get the characteristic information about the DVB-S2X waveform generator.

```
info(s2xWaveGen)
```

```
ans = struct with fields:
              FECFrame: {'normal'  'normal'}
      ModulationScheme: {'QPSK'  '8PSK'}
    LDPCCodeIdentifier: {'9/20'  '25/36'}
```

Create the bit vector of input information bits, `data`, of concatenated TS user packets for each input stream.

```
syncBits = [0 1 0 0 0 1 1 1]';       % Sync byte for TS packet is 47 Hex
pktLen = 1496;                       % UP length without sync bits is 1496
data =  cell(1, s2xWaveGen.NumInputStreams);
for i = 1:s2xWaveGen.NumInputStreams
    numPkts = s2xWaveGen.MinNumPackets(i)*numFrames;
    txRawPkts = randi([0 1], pktLen, numPkts);
    txPkts = [repmat(syncBits, 1, numPkts); txRawPkts];
    data{i} = txPkts(:);
end
```

Generate a DVB-S2X time-domain waveform using the information bits.

```
txWaveform = s2xWaveGen(data);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(s2xWaveGen)
```

```
ans = 40×1 complex

  -0.2412 - 0.0143i
  -0.2619 - 0.0861i
  -0.2726 - 0.1337i
  -0.2511 - 0.1597i
  -0.1851 - 0.1680i
  -0.0780 - 0.1602i
   0.0448 - 0.1288i
   0.1598 - 0.0751i
   0.2482 - 0.0049i
   0.3026 + 0.0702i
      ⋮
```

Get CCSDS TM Waveform Generator Information and Check Transmit Filter Delay

Get information from a `ccsdsTMWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

Create a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) System object. Set the waveform type as `synchronization and channel coding` with low-density parity-check (LDPC) channel coding. Display the properties.

```
tmWaveGen = ccsdsTMWaveformGenerator;
tmWaveGen.WaveformSource = "synchronization and channel coding";
tmWaveGen.ChannelCoding = "LDPC";
tmWaveGen.NumBitsInInformationBlock = 1024;
tmWaveGen.Modulation = "QPSK";
tmWaveGen.CodeRate = "1/2";
disp(tmWaveGen)
```

```
  ccsdsTMWaveformGenerator with properties:

              WaveformSource: "synchronization and channel coding"
               HasRandomizer: true
                      HasASM: true
                   PCMFormat: "NRZ-L"

   Channel coding properties:
                ChannelCoding: "LDPC"
    NumBitsInInformationBlock: 1024
                     CodeRate: "1/2"
                IsLDPCOnSMTF: false

   Digital modulation and filter properties:
                   Modulation: "QPSK"
            PulseShapingFilter: "root raised cosine"
                 RolloffFactor: 0.3500
           FilterSpanInSymbols: 10
              SamplesPerSymbol: 10

  Use get to show all properties
```

Specify the number of transfer frames.

```
numTF = 20;
```

Get the characteristic information about the CCSDS TM waveform generator.

```
info(tmWaveGen)
```

```
ans = struct with fields:
          ActualCodeRate: 0.5000
        NumBitsPerSymbol: 2
    SubcarrierFrequency: []
```

Generate the input bits for the CCSDS TM waveform generator, and then generate the waveform.

```
bits = randi([0 1], tmWaveGen.NumInputBits*numTF,1);
waveform = tmWaveGen(bits);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(tmWaveGen)

ans = 100×1 complex

  -0.0772 - 0.0867i
  -0.0751 - 0.0859i
  -0.0673 - 0.0788i
  -0.0549 - 0.0654i
  -0.0388 - 0.0469i
  -0.0200 - 0.0250i
   0.0002 - 0.0012i
   0.0208 + 0.0227i
   0.0405 + 0.0453i
   0.0587 + 0.0653i
      ⋮
```

Input Arguments

`obj` — Waveform generator

`dvbs2WaveformGenerator` | `dvbs2xWaveformGenerator` | `ccsdsTMWaveformGenerator`

Waveform generator object, specified as a `dvbs2WaveformGenerator`, `dvbs2xWaveformGenerator`, or `ccsdsTMWaveformGenerator` System object.

To enable the `flushFilter` object function when you specify `obj` as a `ccsdsTMWaveformGenerator` System object, you must set these dependencies in the `ccsdsTMWaveformGenerator` object.

- Set the `WaveformSource` property to `"synchronization and channel coding"`.
- Set the `ChannelCoding` property to one of these values.
  - `"none"`
  - `"RS"`
  - `"turbo"`
  - `"LDPC"` — In this case, you must also set the `IsLDPCOnSMTF` property to `0` (`false`)
  - `"convolutional"` — In this case, you must also set the `ConvolutionalCodeRate` property to either `"1/2"` or `"2/3"`
  - `"concatenated"` — In this case, you must also set the `ConvolutionalCodeRate` property to either `"1/2"` or `"2/3"`
- Set the `Modulation` property to either `"BPSK"` or `"QPSK"`.

Output Arguments

`out` — Residual data samples that remain in filter state

column vector

Residual data samples that remain in the filter state, returned as a column vector. The length of the column vector is equal to the product of the `SamplesPerSymbol` and `FilterSpanInSymbols` properties of the input object, `obj`.

When you specify `obj` as `dvbs2WaveformGenerator` or `dvbs2xWaveformGenerator` System object and the `NumInputStream` property as a value greater than 1, the data fields generated from different input streams are merged in a round-robin technique into a single stream. The residual samples of the frame after the merging process are flushed out.

Data Types: `double`

See Also

Functions
`info`

Objects
`ccsdsTMWaveformGenerator` | `dvbs2WaveformGenerator` | `dvbs2xWaveformGenerator`

Introduced in R2021a

satellite

Add satellites to satellite scenario

Syntax

```
satellite(scenario,tlefile)
satellite(scenario,semimajoraxis,eccentricity,inclination,RAAN,
argofperiapsis,trueanomaly)
satellite(scenario,positiontable)
satellite(scenario,positiontable,velocitytable)
satellite(scenario,positiontimeseries)
satellite(scenario,positiontimeseries,velocitytimeseries)
satellite( ___ ,Name,Value)
sat = satellite( ___ )
```

Description

`sat = satellite(scenario,tlefile)` adds a `Satellite` object from TLE file to the satellite scenario specified by `scenario`, specified as a string scalar or character vector. The yaw (z) axes of the satellites point toward nadir, and the roll (x) axes of the satellites align with their respective inertial velocity vectors.

`satellite(scenario,semimajoraxis,eccentricity,inclination,RAAN, argofperiapsis,trueanomaly)` adds a `Satellite` object from Keplerian elements defined in the Geocentric Celestial Reference Frame (GCRF) to the satellite scenario.

`satellite(scenario,positiontable)` adds a `Satellite` object from position data specified in `positiontable` (`timetable` object) to the scenario. This function creates a `Satellite` with `OrbitPropagator="ephemeris"`.

`satellite(scenario,positiontable,velocitytable)` adds a `Satellite` object from position data specified in `positiontable` (`timetable` object) and velocity data specified in `velocitytable` (`timetable` object) to the scenario. This function creates a `Satellite` with `OrbitPropagator="ephemeris"`.

`satellite(scenario,positiontimeseries)` adds a `Satellite` object from position data specified in `positiontimeseries` (`timeseries` object). This function creates a `Satellite` with `OrbitPropagator="ephemeris"`.

`satellite(scenario,positiontimeseries,velocitytimeseries)` adds a `Satellite` object to the `scenario` from position (in meters) data specified in `positiontimeseries` (`timeseries` object) and velocity (in meters/second) data specified in `velocitytimeseries` (`timeseries` object). This function creates a `Satellite` with `OrbitPropagator="ephemeris"`.

`satellite( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to any input argument combination from previous syntaxes. For example, `('Name','satellite1')` specifies the name of the satellite as `'satellite1'`. .

`sat = satellite( ___ )` returns a vector of handles to the added satellites. Specify any input argument combination from previous syntaxes.

Examples

Add Four Satellites from Position Timetable and Visualize Their Trajectories

Add four satellites to the satellite scenario from a position timetable to a satellite scenario and visualize their trajectories.

Create a default satellite scenario object.

```
sc = satelliteScenario;
```

Load a satellite ephemeris timetable, assuming the data is in the GCRF coordinate frame.

```
load("timetableSatelliteTrajectory.mat","positionTT");
```

Add the satellites to the scenario.

```
sat = satellite(sc,positionTT);
```

Visualize the trajectories of the satellites.

```
play(sc);
```

Add Four Satellites from Position and Velocity Timetable and Visualize Their Trajectories

Add four satellites to the satellite scenario from position and velocity timetables in the Earth Centered Earth Fixed (ECEF) frame and visualize their trajectories.

Create a default satellite scenario object.

```
sc = satelliteScenario;
```

Load a satellite ephemeris timetable, assuming the data is in the ECEF coordinate frame.

```
load("timetableSatelliteTrajectory.mat","positionTT","velocityTT");
```

Add the satellites to the scenario.

```
sat = satellite(sc,positionTT,velocityTT,"CoordinateFrame","ecef")
```

Visualize the trajectories of the satellites.

```
play(sc);
```

Add Ground stations to Scenario and Visualize Access Intervals

Create satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020, 5, 1, 11, 36, 0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime, stopTime, sampleTime);
lat = [10];
```

```
lon = [-30];
gs = groundStation(sc, lat, lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
trueAnomaly = 0;
sat = satellite(sc, semiMajorAxis, eccentricity, inclination, ...
        rightAscensionOfAscendingNode, argumentOfPeriapsis, trueAnomaly);
```

Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

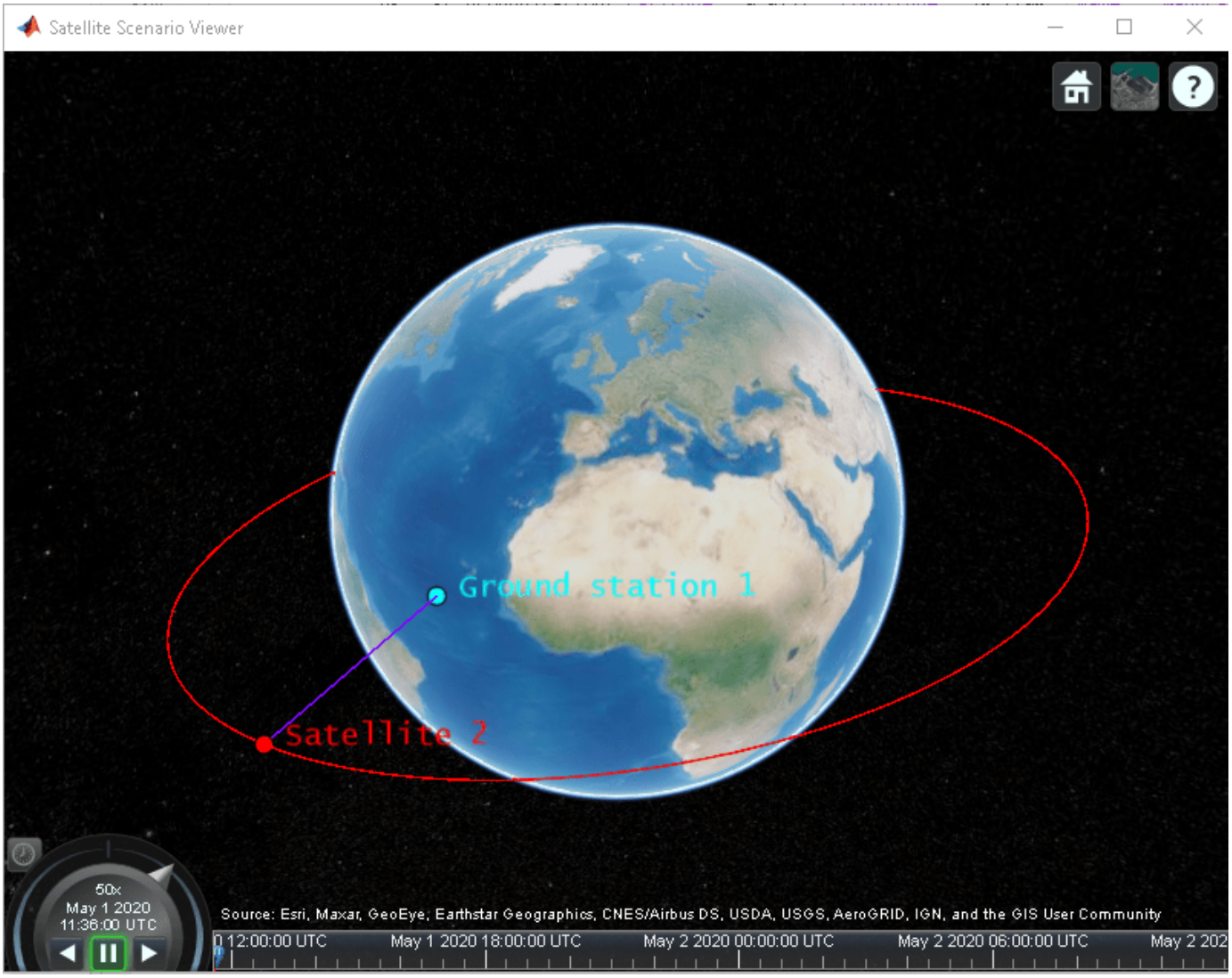
```
ac = access(sat, gs);
intvls = accessIntervals(ac)
```

```
intvls=8×8 table
       Source              Target          IntervalNumber         StartTime                EndTi
    _____________    ___________________    ______________    ____________________    ___________

    "Satellite 2"    "Ground station 1"          1           01-May-2020 11:36:00    01-May-2020
    "Satellite 2"    "Ground station 1"          2           01-May-2020 14:20:00    01-May-2020
    "Satellite 2"    "Ground station 1"          3           01-May-2020 17:27:00    01-May-2020
    "Satellite 2"    "Ground station 1"          4           01-May-2020 20:34:00    01-May-2020
    "Satellite 2"    "Ground station 1"          5           01-May-2020 23:41:00    02-May-2020
    "Satellite 2"    "Ground station 1"          6           02-May-2020 02:50:00    02-May-2020
    "Satellite 2"    "Ground station 1"          7           02-May-2020 05:59:00    02-May-2020
    "Satellite 2"    "Ground station 1"          8           02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



Add Satellites to Scenario Using Keplerian Elements

Create a satellite scenario with a start time of 02-June-2020 8:23:00 AM UTC, and the stop time set to one day later. Set the simulation sample time to 60 seconds.

```
startTime = datetime(2020,6,02,8,23,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add two satellites to the scenario using their Keplerian elements.

```
semiMajorAxis = [10000000; 15000000];
eccentricity = [0.01; 0.02];
inclination = [0; 10];
rightAscensionOfAscendingNode = [0; 15];
```

```
argumentOfPeriapsis = [0; 30];
trueAnomaly = [0; 20];

sat = satellite(sc, semiMajorAxis, eccentricity, inclination, ...
    rightAscensionOfAscendingNode, argumentOfPeriapsis, trueAnomaly)

sat =
  1×2 Satellite array with properties:

    Name
    ID
    ConicalSensors
    Gimbals
    Transmitters
    Receivers
    Accesses
    GroundTrack
    Orbit
    OrbitPropagator
    MarkerColor
    MarkerSize
    ShowLabel
    LabelFontSize
    LabelFontColor
```

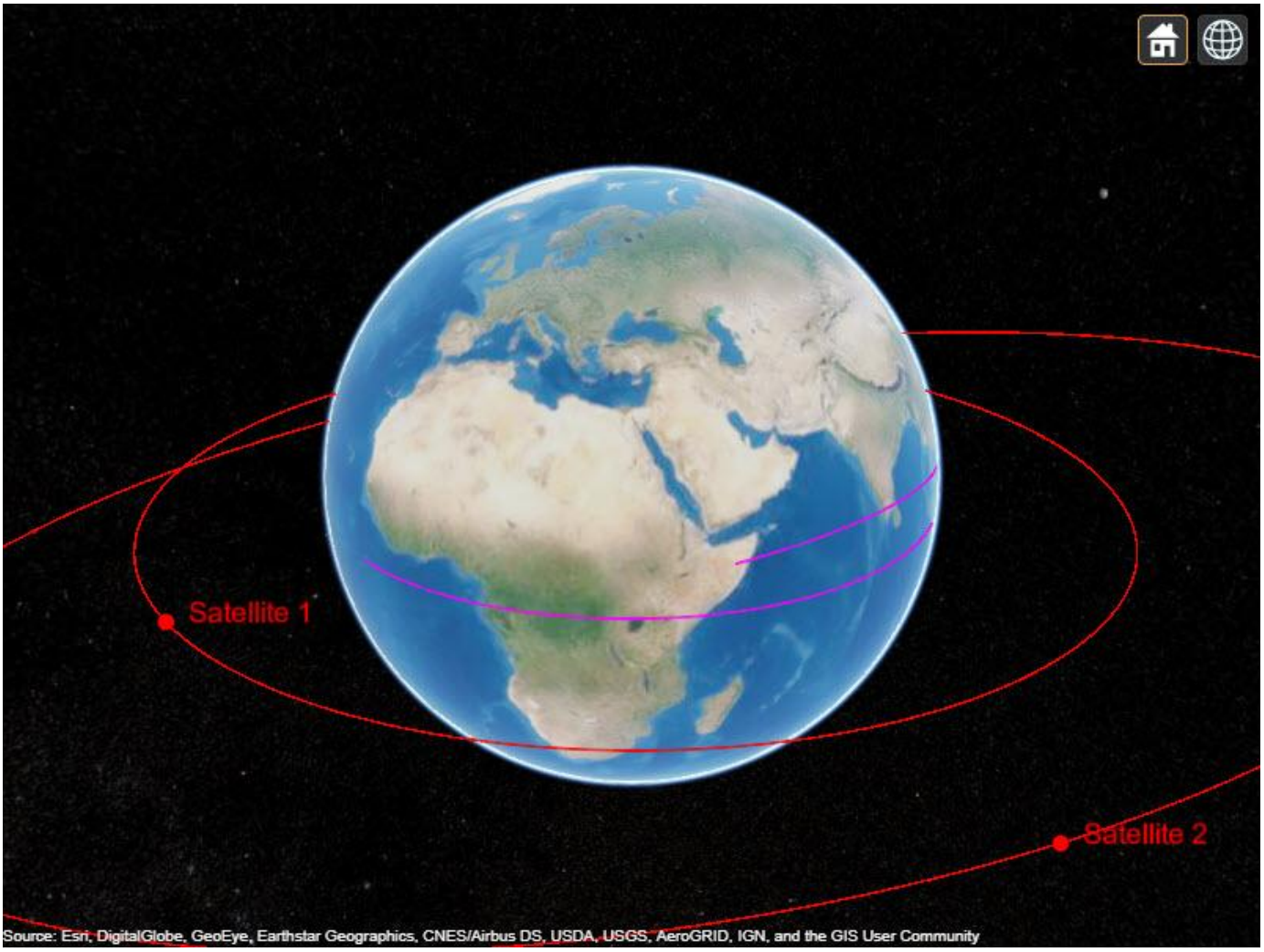
View the satellites in orbit and the ground tracks over one hour.

```
show(sat)
groundTrack(sat,'LeadTime',3600)

ans=1×2 object
  1×2 GroundTrack array with properties:

    LeadTime
    TrailTime
    LineWidth
    TrailLineColor
    LeadLineColor
    VisibilityMode


play(sc)
```



Input Arguments

`scenario` — Satellite scenario

`satelliteScenario` object

Satellite scenario, specified as a `satelliteScenario` object.

`tlefile` — Name of TLE file

character vector | string scalar

Name of a TLE file, specified as a character vector or a string scalar. The TLE file must exist in the current directory, exist in a directory on the MATLAB path, or include a full or relative path to a file.

For more information on TLE files, see “Two Line Element (TLE) Files”.

Data Types: `char` | `string`

`semimajoraxis`, `eccentricity`, `inclination`, `RAAN`, `argofperiapsis`, `trueanomaly` — Keplerian elements defined in GCRF
comma-separated list of vectors

Keplerian elements defined in the GCRF, specified as a comma-separated list of vectors. The Keplerian elements are:

- `semimajoraxis` – This vector defines the semimajor axis of the orbit of the satellite. Each value is equal to half of the longest diameter of the orbit.
- `eccentricity` – This vector defines the shape of the orbit of the satellite.
- `inclination` – This vector defines the angle between the orbital plane and the *xy*-plane of the GCRF for each satellite.
- `RAAN` (right ascension of ascending node) – This element defines the angle between the *xy*-plane of the GCRF and the direction of the ascending node, as seen from the Earth's center of mass for each satellite. The ascending node is the location where the orbit crosses the *xy*-plane of the GCRF and goes above the plane.
- `argofperiapsis` (argument of periapsis) – This vector defines the angle between the direction of the ascending node and the periapsis, as seen from the Earth's center of mass. Periapsis is the location on the orbit that is closest to the Earth's center of mass for each satellite.
- `trueanomaly` – This vector defines the angle between the direction of the periapsis and the current location of the satellite, as seen from the Earth's center of mass for each satellite.

For more information on Keplerian elements, see “Orbital Elements”.

`positiontable` — Position data
timetable | table

Position data in meters, specified as a timetable created using the `timetable` function. `positiontable` has exactly one monotonically increasing column of *rowTimes* (`datetime` or `duration` values) and one or more columns of variables, where each column contains an individual satellite position over time.

If *rowTimes* values are of type `duration`, time values are measured relative to the current scenario `StartTime` property. The timetable `VariableNames` are used by default if no names are provided as an input. Satellite states are assumed to be in the GCRF unless a `CoordinateFrame` name-value argument is provided. States are held constant in GCRF for scenario timesteps outside of the time range of `positiontable`.

Data Types: `table` | `timetable`

`velocitytable` — Velocity data
timetable | table

Velocity data in meters/second, specified as a timetable created using the `timetable` function. `velocitytable` has exactly one monotonically increasing column of *rowTimes* (`datetime` or `duration` values) and one or more columns of variables, where each column contains an individual satellite position over time.

If *rowTimes* values are of type `duration`, time values are measured relative to the current scenario `StartTime` property. The timetable `VariableNames` are used by default if no names are provided as an input. Satellite states are assumed to be in the GCRF unless a `CoordinateFrame` name-value argument is provided. States are held constant in GCRF for scenario timesteps outside of the time range of `velocitytable`.

Data Types: `table` | `timetable`

`positiontimeseries` — Position data
`timeseries` object | `tscollection` object

Position data in meters, specified as a `timeseries` object or a `tscollection` object.

- If the `Data` property of the `timeseries` or `tscollection` object has two dimensions, one dimension must equal 3, and the other dimension must align with the orientation of the time vector.
- If the `Data` property of the `timeseries` or `tscollection` has three dimensions, one dimension must equal 3, either the first or the last dimension must align with the orientation of the time vector, and the remaining dimension defines the number of satellites in the ephemeris.

  When `timeseries.TimeInfo.StartDate` is empty, time values are measured relative to the current scenario `StartTime` property. The timeseries `Name` property (if defined) is used by default if no names are provided as inputs. Satellite states are assumed to be in the GCRF unless a `CoordinateFrame` name-value pair is provided. States are held constant in GCRF for scenario timesteps outside of the time range of `positiontimeseries`.

Data Types: `timeseries` | `tscollection`

`velocitytimeseries` — Velocity data
`timeseries` object | `tscollection` object

Velocity data in meters/second, specified as a `timeseries` object or a `tscollection` object.

- If the `Data` property of the `timeseries` or `tscollection` object has two dimensions, one dimension must equal 3, and the other dimension must align with the orientation of the time vector.
- If the `Data` property of the `timeseries` or `tscollection` has three dimensions, one dimension must equal 3, either the first or the last dimension must align with the orientation of the time vector, and the remaining dimension defines the number of satellites in the ephemeris.

  When `timeseries.TimeInfo.StartDate` is empty, time values are measured relative to the current scenario `StartTime` property. The timeseries `Name` property (if defined) is used by default if no names are provided as inputs. Satellite states are assumed to be in the GCRF unless a `CoordinateFrame` name-value pair is provided. States are held constant in GCRF for scenario timesteps outside of the time range of `velocitytimeseries`.

Data Types: `timeseries` | `tscollection`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Name','MySatellite'` sets the satellite name to `'MySatellite'`.

`Viewer` — Satellite scenario viewer
(default) | row vector of all viewer objects | row vector of `satelliteScenarioViewer` objects

Satellite scenario viewer, specified as a row vector of `satelliteScenarioViewer` objects.

Data Types: `char` | `string`

`Name` — satellite name
`"satellite idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling satellite. After you call satellite, this property is read-only.

satellite name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one satellite is added, specify `Name` as a string scalar or a character vector.
- If multiple satellites are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the satellite added by the `satellite` object function. If another satellite of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`OrbitPropagator` — Name of orbit propagator
`"sgp4"` (default) | `"two-body-keplerian"` | `"sdp4"` | `"ephemeris"`

You can set this property when calling `satellite` only. After you call `satellite`, this property is read-only.

Name of the orbit propagator used for propagating satellite position and velocity, specified as the comma-separated pair consisting of `'OrbitPropagator'` and either `"two-body-keplerian"`, `"sgp4"`, `"sdp4"`, or `"ephemeris"`.

Dependencies

`OrbitPropagator` is not available for ephemeris data inputs (`timetable` or `timeseries`). In these cases, `satellite` ignores this name-value pair.

Data Types: `string` | `char`

`CoordinateFrame` — Satellite state coordinate frame
`"inertial"` (default) | `"ecef"` | `"geographic"`

Satellite state coordinate frame, specified as the comma-separated pair consisting of `'CoordinateFrame'` and one of these values:

- `"inertial"` — For `timeseries` or `timetable` data, specifying this value accepts the position and velocity in the GCRF frame.
- `"ecef"` — For `timeseries` or `timetable` data, specifying this value accepts the position and velocity in the ECEF frame.
- `"geographic"` — For `timeseries` or `timetable` data, specifying this value accepts the position [*lat*, *lon*, *altitude*], where *lat* and *lon* are latitude and longitude in degrees, and *altitude* is the height above the World Geodetic System 84 (WGS 84) ellipsoid in meters.

  Velocity is in the local NED frame.

Dependencies

To enable this name value argument, ephemeris data inputs (`timetable` or `timeseries`).

Data Types: `string` | `char`

Output Arguments

`sat` — Satellite in the scenario

`Satellite` object

Satellite in the scenario, returned as a `Satellite` object belonging to the satellite scenario specified by `scenario`.

You can modify the `Satellite` object by changing its property values.

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`access` | `hide` | `orbitalElements` | `play` | `receiver` | `show` | `transmitter`

Topics

“Multi-Hop Satellite Communications Link Between Two Ground Stations”
“Satellite Constellation Access to a Ground Station”
“Comparison of Orbit Propagators”
“Modeling Satellite Constellations using Ephemeris Data”
“Estimate GNSS Receiver Position with Simulated Satellite Constellations”
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

conicalSensor

Package: satelliteScenario

Add conical sensor to satellite scenario

Syntax

```
conicalSensor(parent)
conicalSensor(parent,Name,Value)
S = conicalSensor( ___ )
```

Description

`conicalSensor(parent)` adds a default `ConicalSensor` object to `parent` which can be a `satellite`, `groundStation` or `gimbal`.

`conicalSensor(parent,Name,Value)` specifies options using one or more name-value arguments. For example, `'MaxViewAngle',90` specifies a field of view angle of 90 degrees.

`S = conicalSensor( ___ )` returns a handle to the added conical sensor. Specify any input argument combination from previous syntaxes.

Input Arguments

`parent` — Element of scenario to which conicalSensor is added

`Satellite` object | `GroundStation` object | `Gimbal` object

Element of scenario to which the conicalSensor is added, specified as a `Satellite`, `GroundStation`, or `Gimbal` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MountingAngle',[20; 35; 10]` sets the yaw, pitch, and roll angles of the conical sensor to 20, 35, and 10 degrees, respectively.

`Name` — conicalSensor name

`"conicalSensor idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling conicalSensor. After you call conicalSensor, this property is read-only.

conicalSensor name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one conicalSensor is added, specify `Name` as a string scalar or a character vector.

- If multiple conicalSensors are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the conicalSensor added by the `conicalSensor` object function. If another conicalSensor of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting location with respect to the parent object, specified as a three-element row vector of positive numbers in meters. The position vector is specified in the body frame of the input `parent`.

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting orientation with respect to parent object, specified as a three-element row vector of positive numbers in degrees. The elements of the vector correspond to yaw, pitch, and roll in that order. Yaw, pitch, and roll are positive rotations about the parent's *z* - axis, intermediate *y* - axis and intermediate *x* - axis of the parent.

Example: `[0; 30; 60]`

`MaxViewAngle` — Field of view angle
`30` (default) | scalar in the range [0, 180]

Field of view angle, specified as a scalar in the range [0, 180]. Units are in degrees.

Output Arguments

`S` — Conical sensor
`ConicalSensor` object

Conical sensor attached to `parent`, returned as a `ConicalSensor` object.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `gimbal` | `groundStation` | `hide` | `play` | `satellite` | `show`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

satelliteScenarioViewer

Create viewer for satellite scenario

Syntax

```
satelliteScenarioViewer(scenario)
satelliteScenarioViewer(scenario,Name,Value)
v = satelliteScenarioViewer(scenario)
```

Description

`satelliteScenarioViewer(scenario)` creates a 3-D or 2-D satellite scenario viewer for the specified satellite scenario.

Note

- Satellite Scenario Viewer is a 3-D map display and requires hardware graphics support for WebGL™.

`satelliteScenarioViewer(scenario,Name,Value)` creates a new viewer using one or more name-value arguments. For example, `'Basemap'`, `'topographic'` bases the scenario on Topographic imagery provided by Esri®.

`v = satelliteScenarioViewer(scenario)` returns the handle to the satellite scenario viewer.

Input Arguments

`scenario` — Satellite scenario

`satelliteScenario` object

Satellite scenario, specified as a `satelliteScenario` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Basemap','topographic'` bases the scenario on Topographic imagery provided by Esri.

`Name` — Name of viewer window

`'Satellite Scenario Viewer'` (default) | string scalar | character vector

Name of the viewer window, specified as a comma-separated pair consisting of `'Name'` and either a string scalar or a character vector.

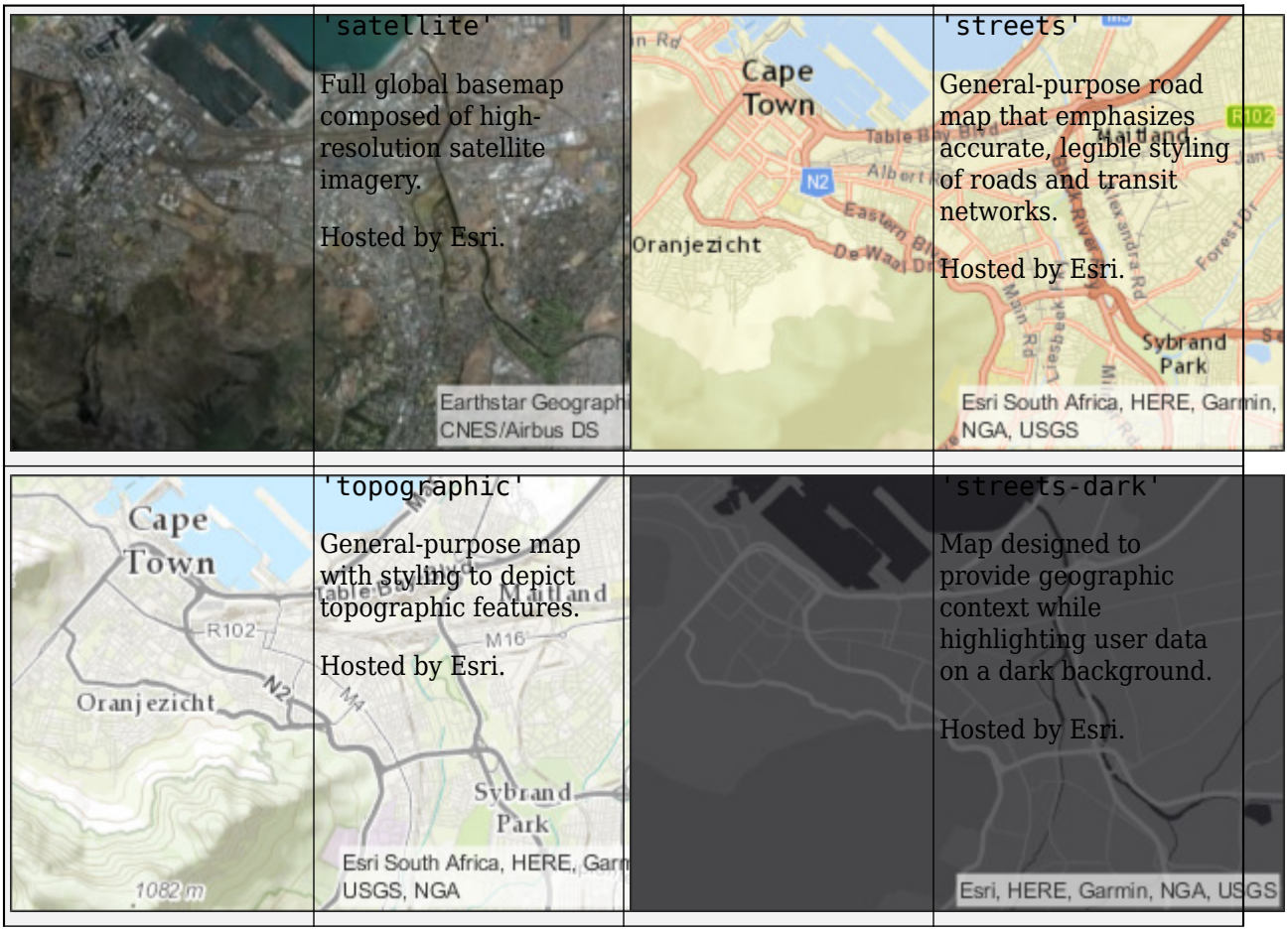
Data Types: `char` | `string`

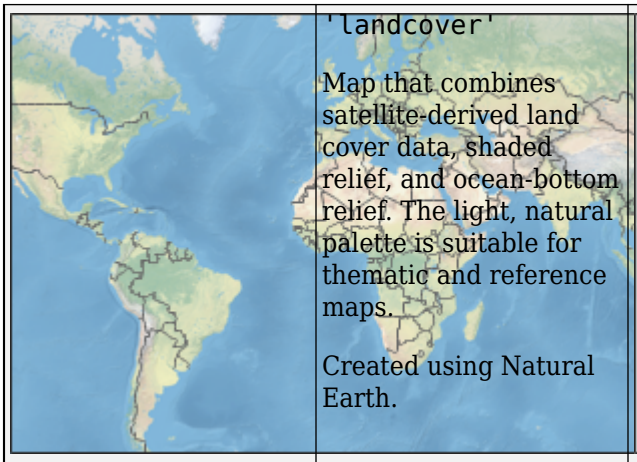
`Position` — Viewer window position
center of the screen (default) | row vector of four elements

Size and location of the satellite scenario window in pixels, specified as a row vector of four elements. The elements of the vector are [left bottom width height]. In the default case, width and height are 800 and 600 pixels, respectively.

`Basemap` — Map on which scenario is visualized
`'satellite'` (default) | `'topographic'` | `'streets'` | `'streets-light'` | `'streets-dark'` | `'darkwater'` | `'grayland'` | `'bluegreen'` | `'colorterrain'` | `'grayterrain'` | `'landcover'`

Map on which scenario is visualized, specified as a comma-separated pair consisting of `'Basemap'` and one of the values specified in this table:

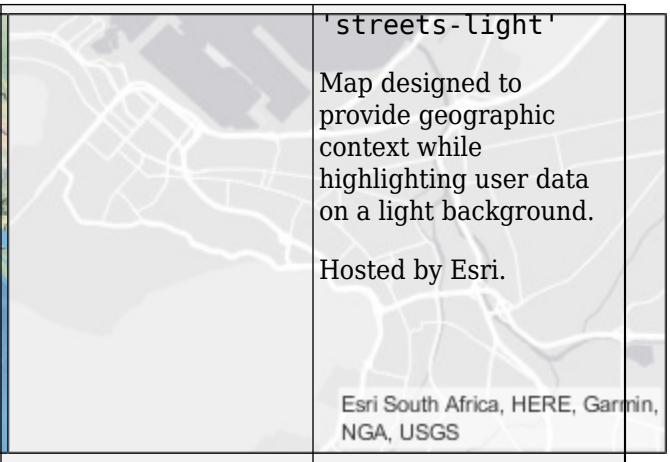




`'landcover'`

Map that combines satellite-derived land cover data, shaded relief, and ocean-bottom relief. The light, natural palette is suitable for thematic and reference maps.

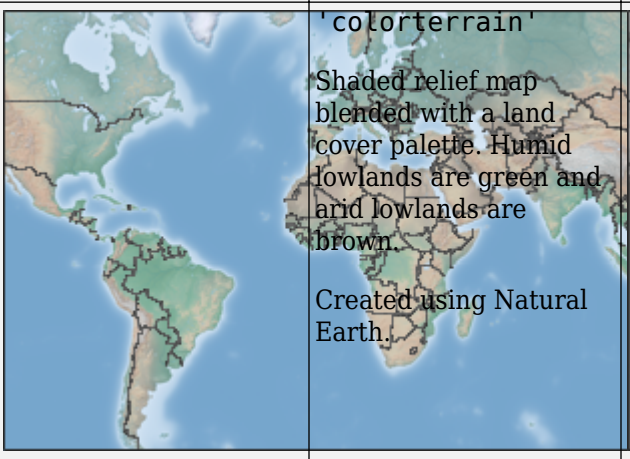
Created using Natural Earth.



`'streets-light'`

Map designed to provide geographic context while highlighting user data on a light background.

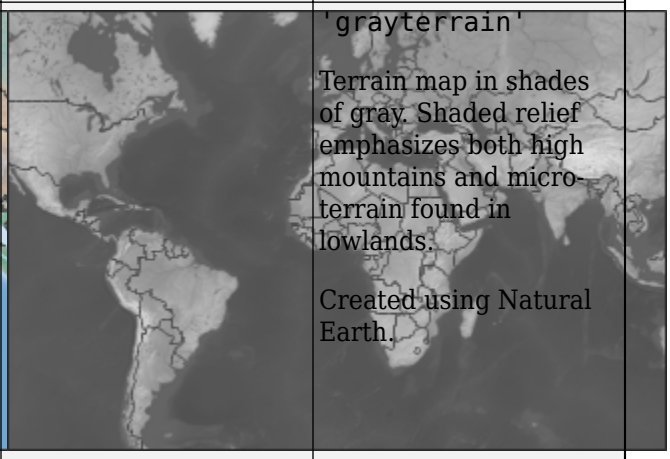
Hosted by Esri.



`'colorterrain'`

Shaded relief map blended with a land cover palette. Humid lowlands are green and arid lowlands are brown.

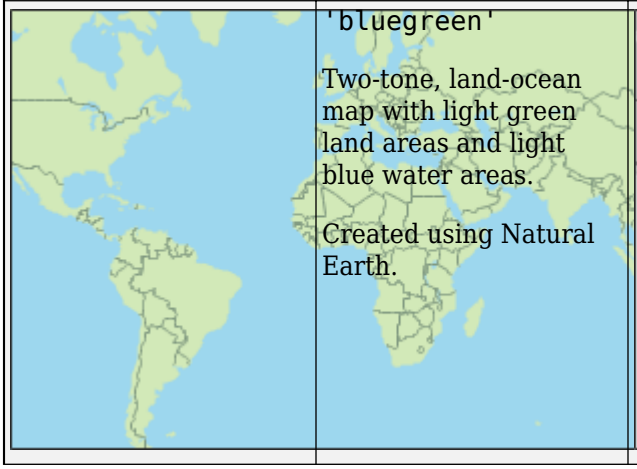
Created using Natural Earth.



`'grayterrain'`

Terrain map in shades of gray. Shaded relief emphasizes both high mountains and micro-terrain found in lowlands.

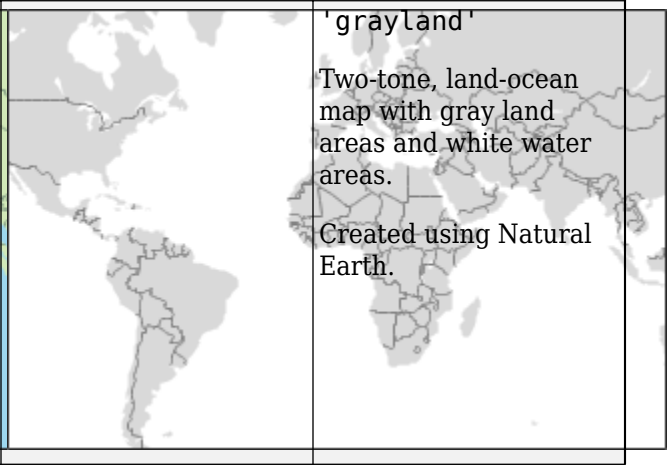
Created using Natural Earth.



`'bluegreen'`

Two-tone, land-ocean map with light green land areas and light blue water areas.

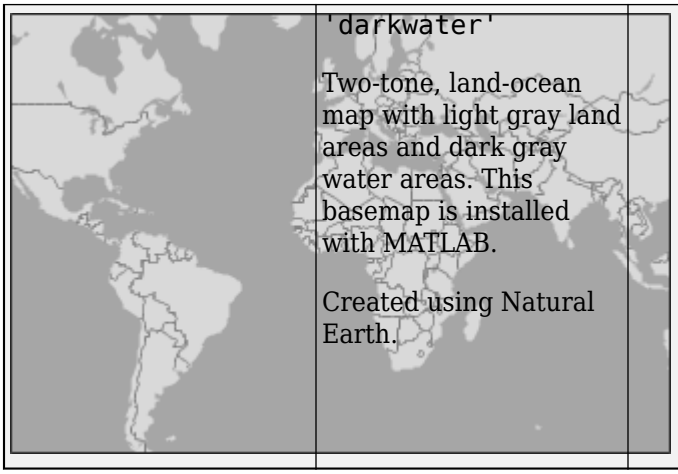
Created using Natural Earth.



`'grayland'`

Two-tone, land-ocean map with gray land areas and white water areas.

Created using Natural Earth.

	`'darkwater'` Two-tone, land-ocean map with light gray land areas and dark gray water areas. This basemap is installed with MATLAB. Created using Natural Earth.		

All basemaps except `'darkwater'` require Internet access. The `'darkwater'` basemap is included with MATLAB and Satellite Communications Toolbox.

If you do not have consistent access to the Internet, you can download the basemaps created using Natural Earth onto your local system by using the Add-On Explorer. The basemaps hosted by Esri are not available for download.

Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by The MathWorks®.

Data Types: `char` | `string`

`Dimension` — Dimension of viewer
`'3-D'` (default) | `'2-D'`

Dimension of the viewer, specified as a comma-separated pair consisting of `'Dimension'` and either `'3-D'` or `'2-D'`.

Data Types: `char` | `string`

`PlaybackSpeedMultiplier` — Speed of animation
`50` (default) | positive scalar

Speed of the animation for the input `scenario` used by the `play` function, specified as a comma-separated pair consisting of `'PlaybackSpeedMultiplier'` and a positive scalar.

`CameraReferenceFrame` — Reference frame of camera
`'ECEF'` (default) | `'Inertial'`

Reference frame of the camera, specified as a comma-separated pair consisting of `'CameraReferenceFrame'` and one of these values:

- `'ECEF'` — Earth-Centered Earth-Fixed camera.
- `'Inertial'` — Inertially fixed camera.

When you specify `'Inertial'`, the globe rotates with respect to the camera. When you specify `'ECEF'`, the camera rotates with the globe.

Dependencies

To enable this name-value argument, set to `Dimension` to `'3-D'`.

`CurrentTime` — Current simulation time

`StartTime` of `satelliteScenario` (default) | `datetime` array

Current simulation time of the viewer, specified as a `datetime` array. This value changes over time when the animation is playing.

Data Types: `datetime`

Output Arguments

v — Satellite scenario viewer

`satelliteScenarioViewer` object

Satellite scenario viewer, returned as a `satelliteScenarioViewer` object.

To specify, query, or visualize satellite scenario viewer details, use these functions:

`campos`	Set or query camera position.
`camheight`	Set or query camera height.
`camheading`	Set or query camera heading angle.
`camroll`	Set or query camera roll angle.
`campitch`	Set or query camera pitch angle.
`camtarget`	Target an object with the camera.
`hideAll`	Hide all visualizations and animations in the Satellite Viewer.
`showAll`	Show all visualizations and animations in the Satellite Viewer.

Tips

- To pan the viewer window without rotation, use **Shift + left click + drag**.

See Also

Functions

`access` | `groundStation` | `hide` | `play` | `satellite` | `show`

Topics

“Multi-Hop Satellite Communications Link Between Two Ground Stations”
“Satellite Constellation Access to a Ground Station”
“Comparison of Orbit Propagators”
“Modeling Satellite Constellations using Ephemeris Data”
“Estimate GNSS Receiver Position with Simulated Satellite Constellations”
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

play

Play satellite scenario simulation results on viewer

Syntax

```
play(scenario)
play(v)
play(scenario,Name,Value)
```

Description

`play(scenario)` plays simulation results of the satellite scenario, `scenario`, from its start time (`StartTime` property) to its stop time (`StopTime` property) using a step size specified by the `SampleTime` property. The function plays the results in a satellite scenario viewer.

`play(v)` plays the satellite scenario simulation on the Satellite Scenario Viewer specified by `v`.

`play(scenario,Name,Value)` specifies additional options using one or more name-value arguments. For example, you can set the speed of animation to 40 times the real time speed, using `'PlaybackSpeedMultiplier',40`.

Examples

Add Satellites to Scenario Using Keplerian Elements

Create a satellite scenario with a start time of 02-June-2020 8:23:00 AM UTC, and the stop time set to one day later. Set the simulation sample time to 60 seconds.

```
startTime = datetime(2020,6,02,8,23,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add two satellites to the scenario using their Keplerian elements.

```
semiMajorAxis = [10000000; 15000000];
eccentricity = [0.01; 0.02];
inclination = [0; 10];
rightAscensionOfAscendingNode = [0; 15];
argumentOfPeriapsis = [0; 30];
trueAnomaly = [0; 20];

sat = satellite(sc, semiMajorAxis, eccentricity, inclination, ...
    rightAscensionOfAscendingNode, argumentOfPeriapsis, trueAnomaly)
```

```
sat =
  1×2 Satellite array with properties:

    Name
    ID
    ConicalSensors
```

```
    Gimbals
    Transmitters
    Receivers
    Accesses
    GroundTrack
    Orbit
    OrbitPropagator
    MarkerColor
    MarkerSize
    ShowLabel
    LabelFontSize
    LabelFontColor
```

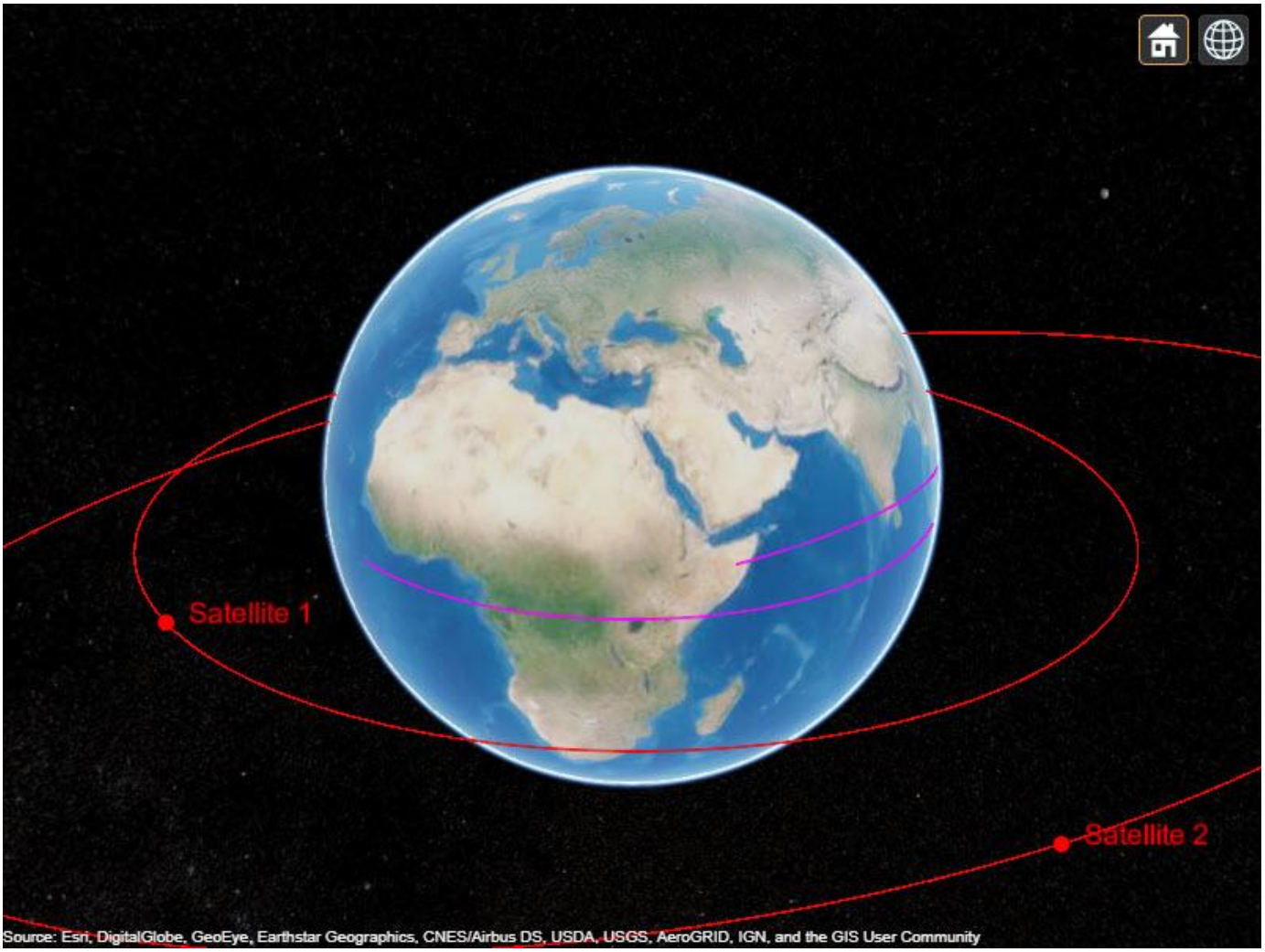
View the satellites in orbit and the ground tracks over one hour.

```
show(sat)
groundTrack(sat,'LeadTime',3600)
```

```
ans=1×2 object
  1×2 GroundTrack array with properties:

    LeadTime
    TrailTime
    LineWidth
    TrailLineColor
    LeadLineColor
    VisibilityMode
```

```
play(sc)
```



Input Arguments

`scenario` — Satellite scenario

`satelliteScenario` object

Satellite scenario, specified as a `satelliteScenario` object.

`v` — Viewer

scalar | vector

Viewer, specified as a scalar or vector of `satelliteScenarioViewer` objects.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'PlaybackSpeedMultiplier',30` plays the animation 30 times faster than real time.

`Viewer` — Viewer on which to play simulation
all viewers associated with `scenario` (default) | `satelliteScenarioViewer` object

Viewer on which to play the simulation, specified as a `satelliteScenarioViewer` object.

`PlaybackSpeedMultiplier` — Speed of animation
`50` (default) | positive scalar

Speed of animation relative to real time, specified as a positive scalar.

See Also

Objects
`satelliteScenario`

Functions
`access` | `groundStation` | `hide` | `satellite` | `show`

Topics
“Multi-Hop Satellite Communications Link Between Two Ground Stations”
“Satellite Constellation Access to a Ground Station”
“Comparison of Orbit Propagators”
“Modeling Satellite Constellations using Ephemeris Data”
“Estimate GNSS Receiver Position with Simulated Satellite Constellations”
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

pointAt

Package: `satelliteScenario`

Target at which entity must be pointed

Syntax

```
pointAt(sat,coordinates)
pointAt(sat,target)
pointAt(sat,'nadir')

pointAt(gim,'none')
pointAt(gim,coordinates)
pointAt(gim,target)
pointAt(gim,'nadir')
```

Description

Satellite Object

`pointAt(sat,coordinates)` sets the attitude of the satellite `sat` such that its yaw (body *z* axis) tracks the geographical coordinates [latitude; longitude; altitude] specified by `coordinates`. The function constantly adjusts the attitude of the satellite so that its yaw (body z) axis points at the desired target. Its roll (body x) axis is aligned with the inertial velocity vector by minimizing the angle between them (exact alignment can be geometrically impossible).

`pointAt(sat,target)` sets the attitude of the satellite `sat` such that its yaw axis tracks the specified `target`.The input `target` can be another satellite or ground station.

`pointAt(sat,'nadir')` sets the attitude of the satellite `sat` such that its yaw axis points in the nadir direction.

Gimbal Object

`pointAt(gim,'none')` sets the steering angles (gimbal azimuth and gimbal elevation) of the gimbal `gim` to zero.

`pointAt(gim,coordinates)` steers `gim` independent of the parent such that its body *z*- axis tracks the geographical coordinates [latitude; longitude; altitude] specified by `coordinates`.

The desired orientation is achieved by rotating the gimbal about its body *z*-axis (gimbal azimuth) and secondly rotating the gimbal about its body *y*-axis (gimbal elevation). The function continuously steers the gimbal for the duration of the simulation so that the gimbal points at the desired target.

`pointAt(gim,target)` steers `gim` such that its body *z*-axis tracks the specified`target`. `target` can be another satellite or ground station.

`pointAt(gim,'nadir')` steers `gim` such that its body *z*-axis points in the nadir direction of the parent, namely, the parent's latitude, longitude, and 0 m altitude.

Input Arguments

`sat` — Satellite

`Satellite` object

Satellite, specified as a `Satellite` object.

`gim` — Gimbal

`Gimbal` object

Gimbal, specified as a `Gimbal` object.

`coordinates` — Geographical coordinates of the satellite target

three-element row vector

Geographical coordinates of the satellite or gimbals' target, specified as a three-element row vector. The latitude and longitude are specified in degrees, and the altitude is specified as the height above the surface of the Earth in meters.

`target` — Target

`Satellite` object | `GroundStation` object

Target at which input `sat` or `gim` is pointed, specified as a `Satellite` or `GroundStation` object.

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`access` | `conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `show` | `transmitter`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

camroll

Package: `satelliteScenario`

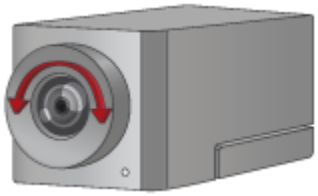
Set or get roll angle of camera for satellite scenario viewer

Syntax

```
camroll(viewer,roll)
outRoll = camroll(viewer,___)
```

Description

`camroll(viewer,roll)` sets the roll angle of the camera for the satellite scenario viewer. Setting the roll angle rotates the camera around its *x*-axis.



`outRoll = camroll(viewer,___)` returns the roll angle of the camera. If the second input is `roll`, then the function sets the output equal to the input roll.

Input Arguments

`viewer` — Satellite scenario viewer

`satelliteScenarioViewer` object

Satellite scenario viewer, specified as a `satelliteScenarioViewer` object. `viewer` must be specified as a scalar `satelliteScenarioViewer` object.[1]

`roll` — Roll angle of camera

scalar in the range [–360, 360]

Roll angle of the camera, specified as a scalar in the range [–360, 360] degrees.

Tips

- When the pitch angle is near –90 (the default value) or 90 degrees, the camera loses one rotational degree of freedom. As a result, when you change the roll angle, the heading angle might change instead. This phenomenon is called gimbal lock. To avoid the effects of gimbal lock, call the `camheading` function instead of the `camroll` function.

1. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks®.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`camheading` | `camheight` | `campitch` | `campos` | `camtarget` | `hide` | `hideAll` | `play` | `show`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

campitch

Package: `satelliteScenario`

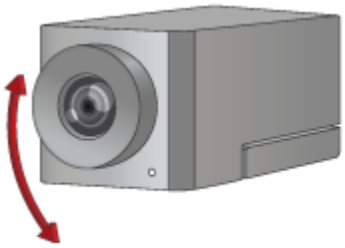
Set or get pitch angle of camera for satellite scenario viewer

Syntax

```
campitch(viewer,pitch)
outPitch = campitch(viewer,___)
```

Description

`campitch(viewer,pitch)` sets the pitch angle of the camera for the specified satellite scenario viewer. Setting the pitch angle tilts the camera up or down as shown in this figure..



`outPitch = campitch(viewer,___)` returns the pitch angle of the camera. If the second input is `pitch`, then the function sets the output equal to the input pitch.

Input Arguments

`viewer` — Satellite scenario viewer

`satelliteScenarioViewer` object

Satellite scenario viewer, specified as a `satelliteScenarioViewer` object. `viewer` must be specified as a scalar `satelliteScenarioViewer` object.[2]

`pitch` — Pitch angle of camera

scalar the in the range [–90, 90]

Pitch angle of the camera, specified as a scalar the in the range [–90, 90] degrees. By default, the pitch angle is –90 degrees, which means that camera points directly toward the surface of the globe.

Tips

- When the pitch angle is near –90 (the default value) or 90 degrees, the camera loses one rotational degree of freedom. As a result, when you change the roll angle, the heading angle might change instead. This phenomenon is called gimbal lock. To avoid the effects of gimbal lock, call the `camheading` function instead of the `camroll` function.

2. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`camheading` | `campitch` | `campos` | `camroll` | `camtarget` | `hide` | `hideAll` | `play` | `show`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

campos

Package: satelliteScenario

Set or get position of camera for satellite scenario viewer

Syntax

```
campos(viewer,lat,lon)
campos(viewer,lat,lon,height)
campos(viewer)
[latOut,lonOut,heightOut] = campos( ___ )
```

Description

`campos(viewer,lat,lon)` sets the latitude and longitude of the camera for the specified satellite scenario viewer.

`campos(viewer,lat,lon,height)` sets the latitude, longitude, and ellipsoidal height of the camera. If you want to set only the height of the camera, use the `camheight` function instead.

`campos(viewer)` displays the latitude, longitude, and ellipsoidal height of the camera as a three-element vector.

`[latOut,lonOut,heightOut] = campos( ___ )` sets the position and then returns the latitude, longitude, and height of the camera. Specify any input argument combinations from previous syntaxes.

Input Arguments

`viewer` — Satellite scenario viewer

`satelliteScenarioViewer` object

Satellite scenario viewer, specified as a `satelliteScenarioViewer` object. `viewer` must be specified as a scalar `satelliteScenarioViewer` object.[3]

`lat` — Geodetic latitude of camera

`0` (default) | scalar in the range [-90, 90].

Geodetic latitude of the camera, specified as a scalar in the range [–90, 90] degrees.

`lon` — Geodetic longitude of camera

`0` (default) | scalar in the range [-360, 360].

Geodetic longitude of the camera, specified as a scalar in the range [–360, 360].

`height` — Ellipsoidal height of camera

`15000000` (default) | numeric scalar

3. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Ellipsoidal height of the camera, specified as a numeric scalar in meters. Satellite scenario viewer objects use the WGS84 reference ellipsoid.

If you specify the height so that the camera is level with or below the surface of the Earth, then the campos function sets the height to a value one meter above the surface.

Output Arguments

`latOut` — Geodetic latitude of camera

numeric scalar

Geodetic latitude of the camera, returned as a numeric scalar in degrees.

`lonOut` — Geodetic longitude of camera

numeric scalar

Geodetic longitude of the camera, returned as a numeric scalar in degrees.

`heightOut` — Ellipsoidal height of camera

numeric scalar

Ellipsoidal height of the camera, returned as a numeric scalar in meters. Satellite scenario viewer objects use the WGS84 reference ellipsoid. For more information about ellipsoidal height, see “Geodetic Coordinates”.

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`camheading` | `camheight` | `campitch` | `camroll` | `camtarget` | `hide` | `hideAll` | `play` | `show`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

camheading

Package: `satelliteScenario`

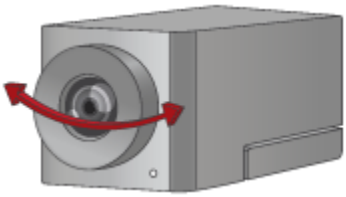
Set or get heading angle of camera for satellite scenario satellite scenario viewer

Syntax

```
camheading(viewer,heading)
outHeading = camheading(viewer, ___ )
```

Description

`camheading(viewer,heading)` sets the heading angle of the camera for the specified satellite scenario viewer. Setting the heading angle shifts the camera left or right about its z - axis.



`outHeading = camheading(viewer, ___ )` returns the heading angle of the camera. If the second input is `heading`, then the function sets the output equal to the input pitch.

Input Arguments

`viewer` — Satellite scenario viewer

`satelliteScenarioViewer` object

Satellite scenario viewer, specified as a `satelliteScenarioViewer` object. `viewer` must be specified as a scalar `satelliteScenarioViewer` object.[4]

`heading` — Heading angle of camera

`360` (default) | scalar in the range [–360, 360]

Heading angle of the camera, specified as a scalar value in the range [–360, 360] degrees.

Tips

- When the pitch angle is near –90 (the default value) or 90 degrees, the camera loses one rotational degree of freedom. As a result, when you change the roll angle, the heading angle might change instead. This phenomenon is called gimbal lock. To avoid the effects of gimbal lock, call the `camheading` function instead of the `camroll` function.

4. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`camheading` | `camheight` | `campitch` | `campos` | `camroll` | `camtarget` | `hide` | `hideAll` | `play` | `show`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

camheight

Package: satelliteScenario

Set or get height of camera for satellite scenario viewer

Syntax

```
camheight(viewer,height)
heightOut = camheight(viewer, ___ )
```

Description

`camheight(viewer,height)` sets the ellipsoidal height of the camera for the specified satellite scenario viewer.

`heightOut = camheight(viewer, ___ )` returns the ellipsoidal height of the camera. If the second input is `height`, then the function sets the output equal to the input height.

Input Arguments

`viewer` — Satellite scenario viewer

`satelliteScenarioViewer` object

Satellite scenario viewer, specified as a `satelliteScenarioViewer` object. `viewer` must be specified as a scalar `satelliteScenarioViewer` object.[5]

`height` — Ellipsoidal height of camera

`15000000` (default) | numeric scalar

Ellipsoidal height of the camera, specified as a numeric scalar in meters. Satellite scenario viewer objects use the WGS84 reference ellipsoid. For more information about ellipsoidal height, see “Geodetic Coordinates”.

If you specify the height so that the camera is level with or below the surface of the Earth, then the `camheight` function sets the height to a value one meter above the surface.

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`camheading` | `campitch` | `campos` | `camroll` | `camtarget` | `hide` | `hideAll` | `play` | `show`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”

5. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

“Satellite Scenario Basics”

Introduced in R2021a

camtarget

Package: satelliteScenario

Set camera target for satellite scenario viewer

Syntax

```
camtarget(viewer,target)
```

Description

`camtarget(viewer,target)` focuses the camera on the input satellite or ground station. The camera follows the object and can be unlocked by calling `camtarget` on another satellite or ground station or by double-clicking anywhere in the map.

Input Arguments

`viewer` — Satellite scenario viewer

`satelliteScenarioViewer` object

Satellite scenario viewer, specified as a `satelliteScenarioViewer` object. `viewer` must be specified as a scalar `satelliteScenarioViewer` object.[6]

`target` — Target of camera

`Satellite` object | `GroundStation` object

Target of the camera, specified as a scalar `Satellite` or `GroundStation` object.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`camheading` | `camheight` | `campitch` | `campos` | `camroll` | `hide` | `hideAll` | `play` | `show`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

6. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

hideAll

Package: `satelliteScenario`

Hide all graphics in satellite scenario viewer

Syntax

```
hideAll(viewer)
```

Description

`hideAll(viewer)` hides all graphics in the specified satellite scenario viewer.

Input Arguments

`viewer` — Satellite scenario viewer

`satelliteScenarioViewer` object

Satellite scenario viewer, specified as a `satelliteScenarioViewer` object. `viewer` must be specified as a scalar `satelliteScenarioViewer` object.[7]

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`access` | `camheading` | `camheight` | `campitch` | `campos` | `camroll` | `camtarget` | `conicalSensor` | `groundStation` | `hide` | `play` | `show` | `showAll`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”
“Comparison of Orbit Propagators”

Introduced in R2021a

7. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

showAll

Package: satelliteScenario

Show all graphics in viewer

Syntax

```
showAll(viewer)
```

Description

`showAll(viewer)` shows all graphics in the specified satellite scenario viewer.

Input Arguments

viewer — Satellite scenario viewer

satelliteScenarioViewer object

Satellite scenario viewer, specified as a `satelliteScenarioViewer` object. `viewer` must be specified as a scalar `satelliteScenarioViewer` object.[8]

See Also

Objects

`access` | `conicalSensor` | `groundStation` | `satelliteScenario` | `satelliteScenarioViewer`

Functions

`camheading` | `camheight` | `campitch` | `campos` | `camroll` | `camtarget` | `hide` | `play` | `show`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”
“Comparison of Orbit Propagators”

Introduced in R2021a

8. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

accessPercentage

Package: `satelliteScenario`

Percentage of time when access exists between first and last node defining access analysis

Syntax

```
ap = accessPercentage(ac)
```

Description

`ap = accessPercentage(ac)` returns the percentages of time from start time to stop time of the satellite scenario when access exists between the first and last node of each access object in the input vector.

Input Arguments

ac — Access analysis

row vector of `Access` objects

Access analysis, specified as a row vector of a `Access` objects.

Outputs Arguments

ap — Access percentage

row vector of nonnegative numbers

Access percentage, returned as a row vector of nonnegative numbers.

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `show` | `transmitter`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

linkPercentage

Package: `satelliteScenario`

Percentage of time when link between first and last node in link analysis is closed

Syntax

```
lp = linkPercentage(lnk)
```

Description

`lp = linkPercentage(lnk)` returns the percentages of time from start time to stop time of the satellite scenario when link between the first and last node is closed.

Input Arguments

`lnk` — Link analysis

`Link` object scalar

Link analysis object, specified as a `Link` object scalar.

Outputs Arguments

`lp` — Link percentage

vector of positive numbers

Link percentage, returned as a vector of positive numbers.

See Also

Objects

`Link` | `satelliteScenario` | `satelliteScenarioViewer`

Functions

`ebno` | `groundStation` | `linkIntervals` | `linkStatus` | `play` | `show`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

linkStatus

Package: `satelliteScenario`

Status of link closure between first and last node

Syntax

```
s = linkStatus(lnk)
s = linkStatus(lnk,timeIn)
[s,timeOut] = linkStatus( ___ )
```

Description

`s = linkStatus(lnk)` returns the link closure status history between the first and last node in the input Link object.

`s = linkStatus(lnk,timeIn)` returns the link closure status at the specified datetime in `timeIn`.

`[s,timeOut] = linkStatus( ___ )` returns the link closure status and the corresponding times in Universal Time Coordinated (UTC).

Input Arguments

`lnk` — Link analysis

`Link` object scalar

Link analysis object, specified as a `Link` object scalar.

`timeIn` — Time at which output is calculated

scalar

Time at which the output is calculated, specified as a scalar. If you do not specify a time zone, then the time zone is assumed to be UTC.

Outputs Arguments

`s` — Link closure status

scalar or row vector of logical values

Link closure status, returned as a row vector of logical values. If `timeIn` is specified, `s` is a row vector, otherwise, the output is a scalar. The status at a given instant is `1` (`true`) if the link between first and last node is closed. The link between the first and last node is closed when the link between each individual pair of intermediate adjacent nodes in the Sequence property of the link is closed.

- For a given pair, the link is considered to be closed when both nodes belong to the same satellite or ground station.
- Otherwise, the link between the pair is closed if the directionality is from a transmitter to a receiver and the energy per bit to noise power spectral density ratio (Eb/No) at the receiver is greater than its `RequiredEbNo`.

- Additionally, if a given node is attached to a ground station directly or via a gimbal, the elevation angle of the adjacent node with respect to the ground station must be greater than or equal to its MinElevationAngle.

`timeOut` — Time samples of output link status
scalar | vector

Time samples of output link status, returned as a scalar or a vector. If time history of link status is returned, `timeOut` is a row vector.

See Also

Objects
`Link` | `groundStation` | `satelliteScenario` | `satelliteScenarioViewer`

Functions
`ebno` | `linkIntervals` | `linkPercentage` | `play` | `show`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

linkIntervals

Package: `satelliteScenario`

Intervals during which link is closed

Syntax

```
int = linkIntervals(lnk)
```

Description

`int = linkIntervals(lnk)` returns a table of intervals during which the link between the first node and last node in each link object input vector is closed.

Input Arguments

`lnk` — Link analysis

`Link` object scalar

Link analysis object, specified as a `Link` object scalar.

Outputs Arguments

`int` — Intervals during which link is closed

table

Intervals during which the link is closed, returned as a table.

Each row of the table denotes a specific interval, and the columns of the table are named `Source`, `Target`, `IntervalNumber`, `StartTime`, `EndTime`, `Duration` (in seconds), `StartOrbit`, and `EndOrbit`. `Source` and `Target` are the names of the first and last node, respectively, that define the link analysis.

- If `Source` is directly or indirectly attached to a satellite, then `StartOrbit` and `EndOrbit` correspond to the satellite associated with `Source`.
- If `Target` is directly or indirectly attached to a satellite, then `StartOrbit` and `EndOrbit` correspond to the satellite associated with the `Target`. Otherwise, `StartOrbit` and `EndOrbit` are `NaN` because they are associated with ground stations.

See Also

Objects

`Link` | `groundStation` | `satelliteScenario` | `satelliteScenarioViewer`

Functions

`ebno` | `linkPercentage` | `linkStatus` | `play` | `show`

Topics

“Model, Visualize, and Analyze Satellite Scenario”

“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

aer

Package: `satelliteScenario`

Calculate azimuth angle, elevation angle, and range in NED frame from another satellite or ground station

Syntax

```
az = aer(objIn,target)
[az,el] = aer(objIn,target)
[az,el,range] = aer(objIn,target)
[az,el,range,timeOut] = aer(objIn,target)
[___] = aer(objIn,target,timeIn)
```

Description

`az = aer(objIn,target)` returns the history of azimuth angles, between satellite or ground station `objIn` and another satellite or ground station `target` belonging to a given `satelliteScenario` object.

`[az,el] = aer(objIn,target)` returns the history of elevation angles, `el`, between satellite or ground station `objIn` and another satellite or ground station `target`.

`[az,el,range] = aer(objIn,target)` returns the history of the `range` of `target` with respect to `objIn`.

`[az,el,range,timeOut] = aer(objIn,target)` returns the corresponding time in `timeOut`.

`[___] = aer(objIn,target,timeIn)` returns the outputs at the specified datetime `timeIn`. Specify any output argument combinations from previous syntaxes.

Input Arguments

`objIn` — First scenario component

`Satellite` object | `GroundStation` object

First scenario component, specified as a `Satellite` or `GroundStation` object.

`target` — Second scenario component

`Satellite` object | `GroundStation` object

Second scenario component, specified as a `Satellite` or `GroundStation` object.

`timeIn` — Time at which output is calculated

scalar | vector | matrix | N-D array

Time at which output is calculated, returned as a scalar, vector, matrix, or N-D array. If no time zone is specified in `timeIn`, the time zone is assumed to be UTC.

Output Arguments

az — Azimuth angles

scalar | vector

Azimuth angles of target in the local azimuth, elevation and range (AER) system, returned as a scalar or vector. Azimuths are measured clockwise from North. Values are specified in degrees in the interval [0, 360). The vector elements correspond to the time samples from the satellite scenario StartTime to StopTime properties, as specified by the SampleTime property. The azimuth angle is defined in the North-East-Down (NED) frame of (and centered at) `objIn` such that 0 degrees is North, 90 degrees is East, 180 degrees is South, and 270 degrees is West.

el — Elevation angles

scalar | vector

Elevation angles of target in the local AER system, returned as a scalar or vector. Elevations are measured with respect to a plane that is perpendicular to the normal of the surface of the earth. If `objIn` is on the surface of the Earth, then the plane is tangent to the Earth.

Values are specified in degrees in the closed interval [0 180]. The vector elements correspond to the time samples from the satellite scenario StartTime to StopTime properties, as specified by the SampleTime property. The elevation angle is defined in the NED frame of (and centered at) `objIn` such that 0 deg implies `target` is on the North East (NE) plane, 90 degrees implies `target` is directly above `objIn`, and -90 degrees implies `target` is directly below `objIn`.

range — Distances from local origin

scalar | vector

Distances from the local origin in meters, returned as a scalar or vector.

timeOut — Time samples between start and stop time of scenario

scalar | vector

Time samples corresponding to `az`, `el`, and `range`, returned as a scalar or vector.

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`access` | `conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `show` | `transmitter`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

accessIntervals

Package: `satelliteScenario`

Intervals during which access status is true

Syntax

```
int = accessIntervals(ac)
```

Description

`int = accessIntervals(ac)` returns a table of intervals during which the access status corresponding to each access object in the input vector is true.

Examples

Add Ground stations to Scenario and Visualize Access Intervals

Create satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020, 5, 1, 11, 36, 0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime, stopTime, sampleTime);
lat = [10];
lon = [-30];
gs = groundStation(sc, lat, lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
trueAnomaly = 0;
sat = satellite(sc, semiMajorAxis, eccentricity, inclination, ...
        rightAscensionOfAscendingNode, argumentOfPeriapsis, trueAnomaly);
```

Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

```
ac = access(sat, gs);
intvls = accessIntervals(ac)
```

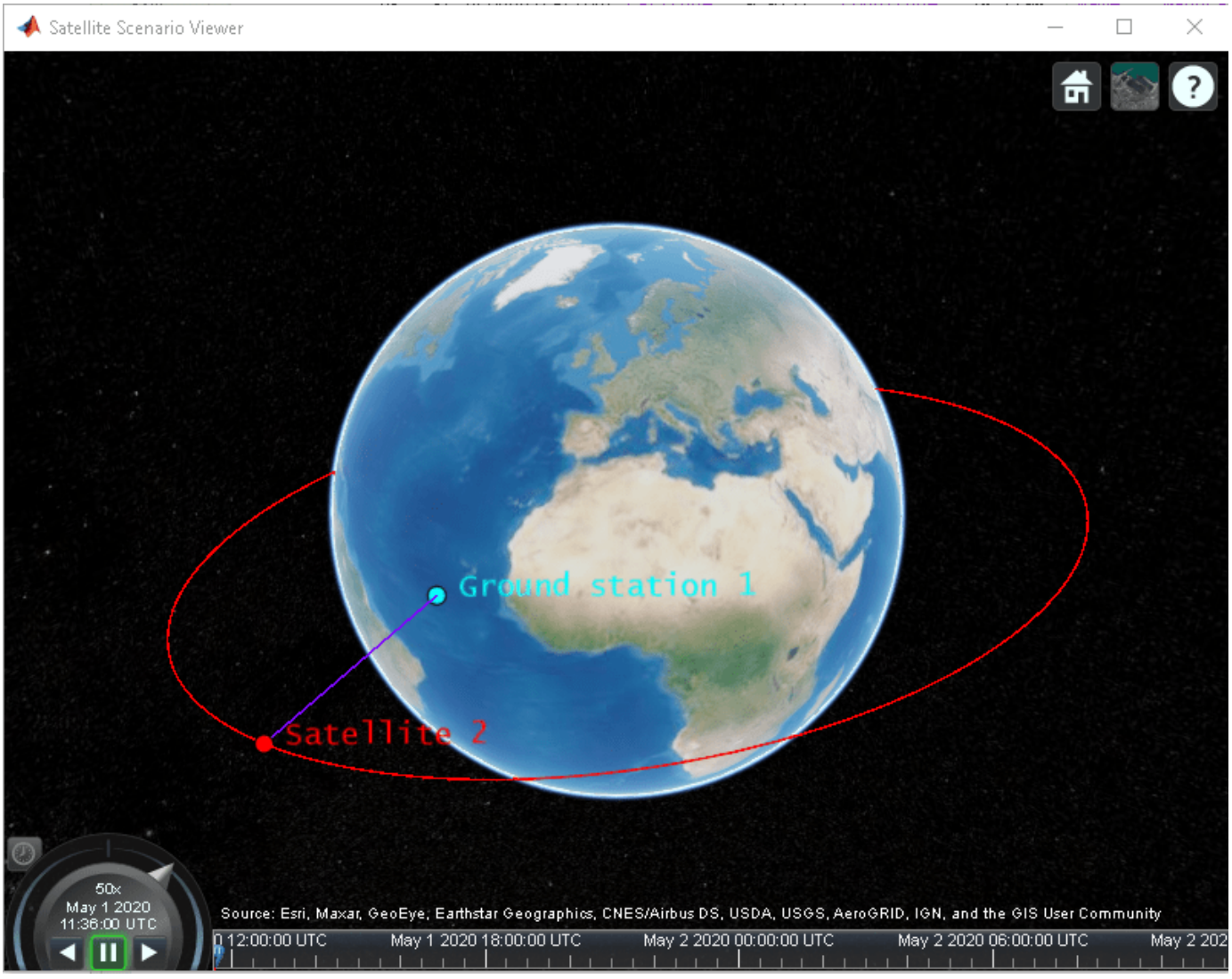
```
intvls=8×8 table
       Source              Target          IntervalNumber         StartTime                EndTi
    _____________    ___________________    ______________    ____________________    ___________

    "Satellite 2"    "Ground station 1"          1           01-May-2020 11:36:00    01-May-2020
    "Satellite 2"    "Ground station 1"          2           01-May-2020 14:20:00    01-May-2020
```

```
"Satellite 2"    "Ground station 1"          3          01-May-2020 17:27:00    01-May-2020
"Satellite 2"    "Ground station 1"          4          01-May-2020 20:34:00    01-May-2020
"Satellite 2"    "Ground station 1"          5          01-May-2020 23:41:00    02-May-2020
"Satellite 2"    "Ground station 1"          6          02-May-2020 02:50:00    02-May-2020
"Satellite 2"    "Ground station 1"          7          02-May-2020 05:59:00    02-May-2020
"Satellite 2"    "Ground station 1"          8          02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



Input Arguments

ac — Access analysis

row vector of `Access` objects

Access analysis, specified as a row vector of a `Access` objects.

Outputs Arguments

`int` — Intervals during which access is true

table

Intervals during which access is true, returned as a table.

Each row of the table denotes a specific interval, and the columns of the table are named `Source`, `Target`, `IntervalNumber`, `StartTime`, `EndTime`, `Duration` (in seconds), `StartOrbit`, and `EndOrbit`. `Source` and `Target` are the names of the first and last node, respectively, defining the access analysis.

- If `Source` is a satellite or an object that is directly or indirectly attached to a satellite, then `StartOrbit` and `EndOrbit` correspond to the satellite associated with Source.
- If `Target` is a satellite or an object that is directly or indirectly attached to a satellite, then `StartOrbit` and `EndOrbit` correspond to the satellite associated with `Target`. Otherwise, `StartOrbit` and `EndOrbit` are `NaN` because they are associated with ground stations.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `show` | `transmitter`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

orbitalElements

Package: `satelliteScenario`

Orbital elements of satellites in scenario

Syntax

```
elements = orbitalElements(sat)
```

Description

`elements = orbitalElements(sat)` returns the orbital elements of the specified satellite `sat`.

Input Arguments

`sat` — Satellite

row vector of `Satellite` objects

Satellite, specified as a row vector of `Satellite` objects.

Output Arguments

`elements` — Orbital elements

structure

Orbital elements of input `sat`, returned as a structure. The fields of the structure depend on the orbit propagator chosen using the OrbitPropagator property of the `satelliteScenario` object.

For more information regarding orbital elements, see “Orbital Elements”.

Two Body Keplerian

The two-body-keplerian orbit propagator has these fields:

- `SemiMajorAxis`
- `Eccentricity`
- `Inclination`
- `RightAscensionOfAscendingNode`
- `ArgumentOfPeriapsis`
- `TrueAnomaly`
- `Period`

SGP4 and SDP4

The sgp4 and sdp4 orbit propagators have these fields:

- `Eccentricity`

- `Inclination`
- `RightAscensionOfAscendingNode`
- `ArgumentOfPeriapsis`
- `MeanAnomaly`
- `MeanMotion`
- `Epoch`
- `BStar`
- `Period`

The orbital elements represent the mean values at `Epoch`.

Ephemeris

The ephemeris propagator has these fields:

- `EphemerisStartTime`
- `EphemerisStopTime`
- `PositionTimeTable`
- `VelocityTimeTable`

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `conicalSensor` | `groundStation` | `play` | `receiver` | `satellite` | `show` | `transmitter`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

accessStatus

Package: satelliteScenario

Status of access between first and last node defining access analysis

Syntax

```
s = accessStatus(ac)
s = accessStatus(ac,timeIn)
[s,timeOut] = accessStatus( ___ )
```

Description

`s = accessStatus(ac)` returns the access status history between the first and last node defining each access object in the input vector.

`s = accessStatus(ac,timeIn)` returns the status of each access analysis object at the specified datetime in `timeIn`.

`[s,timeOut] = accessStatus( ___ )` returns the status of each access analysis object and the corresponding datetime in Universal Time Coordinated (UTC).

Input Arguments

ac — Access analysis

row vector of `Access` objects

Access analysis, specified as a row vector of `Access` objects.

timeIn — Time at which output is calculated

scalar

Time at which the output is calculated, specified as a scalar. If you do not specify a time zone, then the time zone is assumed to be UTC.

Outputs Arguments

s — Access analysis status

scalar or row vector of `logical` values

Access analysis status, returned as a scalar or row vector of `logical` values. If `timeIn` is specified, `s` is a row vector, otherwise, the output is a scalar. The status at a given instant is `1` (`true`) if access exists between each pair of adjacent nodes defined by `Sequence`. For example, in a given pair, say defined by node1 and node2, node1 has access to node2 and vice versa.

- If a node is a satellite, then the satellite has access to the adjacent node if both nodes are in line of sight of each other.

- If a node is a ground station, then the ground station has access to the adjacent node if the elevation angle of the node with respect to the ground station is greater than or equal to the MinElevationAngle property of `GroundStation`.
- If a node is a conical sensor, then the conical sensor has access to the adjacent node if the latter is in the field of view of the former. If the conical sensor is attached to a ground station directly or via a gimbal, then the elevation angle of the adjacent node with respect to the ground station must be greater than or equal to the MinElevationAngle property of `GroundStation`.

`timeOut` — Time samples of output access status
scalar | vector

Time samples of the output access status, returned as a scalar or vector. If the time history of the access status is returned, `timeOut` is a row vector.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `show` | `transmitter`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

states

Package: `satelliteScenario`

Position and velocity of satellite

Syntax

```
pos = states(sat)
[pos,velocity] = states(sat)
[___] = states(sat,timeIn)
[___] = states(___,'CoordinateFrame',C)
[pos,velocity,timeOut] = states(___)
```

Description

`pos = states(sat)` returns a 3-by-*n* matrix with the position history of the satellite `sat` in the Geocentric Celestial Reference Frame (GCRF), where *n* is the number of time samples in the satellite scenario simulation.

`[pos,velocity] = states(sat)` returns a 3-by-*n* matrix with the position and velocity history of satellite in GCRF.

`[___] = states(sat,timeIn)` also returns the outputs at the times specified by `timeIn`. Specify any output argument combinations from previous syntaxes.

`[___] = states(___,'CoordinateFrame',C)` returns the outputs in the coordinates specified by `C`.

`[pos,velocity,timeOut] = states(___)` returns the position and velocity history of the satellite and the corresponding time in Universal Time Coordinated (UTC).

Input Arguments

`sat` — Satellite

row vector of `Satellite` objects

Satellite, specified as a row vector of `Satellite` objects.

`timeIn` — Time at which output is calculated

scalar

Time at which the output is calculated, specified as a scalar. If you do not specify a time zone, then the time zone is assumed to be UTC.

`C` — Coordinate frame

`'ecef'` | `'inertial'` | `'geographical'`

Coordinate frame in which the outputs are returned, specified as `'ecef'`, `'inertial'`, or `'geographical'`.

- The `'ecef'` option returns the coordinates in the Earth Centered Earth Fixed (ECEF) frame. For more information on ECEF frames, see “Earth-Centered Earth-Fixed Coordinates”.
- The `'inertial'` option returns the coordinates in the GCRF frame.
- The `'geographic'` option returns the position as [*lat*; *lon*; *altitude*], where *lat* and *lon* are latitude and longitude in degrees, and altitude is the height above the wgs84 ellipsoid in meters. The velocity returned is ECEF, defined in the local North-East-Down (NED) frame.

Output Arguments

`pos` — Position history

scalar | vector | matrix | *N*-D array

Position history of the satellite, returned as a scalar, vector, matrix, or *N*-D array in the GCRF frame. Units are in meters.

`velocity` — Velocity history

scalar | vector | matrix | *N*-D array

Velocity history of the satellite, returned as a scalar, vector, matrix, or *N*-D array in the GCRF frame. Units are in meters/second.

`timeOut` — Time samples of position and velocity

scalar | vector | matrix | *N*-D array

Time samples of the position and velocity of the satellite, returned as a scalar, vector, matrix, or *N*-D array. If time histories of the position and velocity of the satellite are returned, `timeOut` is a row vector.

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`access` | `groundStation` | `hide` | `play` | `show`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

gimbalAngles

Steering angles of gimbal

Syntax

```
az = gimbalAngles(gim)
[az,el] = gimbalAngles(gim)
[az,el,timeOut] = gimbalAngles(gim)
[___] = gimbalAngles(gim,timeIn)
```

Description

`az = gimbalAngles(gim)` returns the steering azimuth of the specified gimbal., in degrees. The gimbal is steered to the desired pointing direction by first rotating it about its body z - axis (steering azimuth) and secondly rotating it about its body y - axis (steering elevation).

`[az,el] = gimbalAngles(gim)` returns the steering azimuth and steering elevation of the specified gimbal.

`[az,el,timeOut] = gimbalAngles(gim)` returns the steering azimuth and steering elevation of the gimbal and the corresponding time in UTC.

`[___] = gimbalAngles(gim,timeIn)` returns the steering azimuth and steering elevation (depending on the specified output arguments) of the gimbal at the specified time. If you do not specify a time zone, the time zone is assumed to be Universal Time Coordinated (UTC).

Input Arguments

`gim` — Gimbal

`Gimbal` object

Gimbal whose steering angle is being calculated, specified as a `Gimbal` object.

Output Arguments

`az` — Azimuth angles

scalar | vector

Azimuth angles of target in the local azimuth, elevation and range (AER) system, returned as a scalar or vector. Azimuths are measured clockwise from North. Values are specified in degrees in the interval [0, 360). The vector elements correspond to the time samples from the satellite scenario StartTime to StopTime properties, as specified by the SampleTime property. The azimuth angle is defined in the North-East-Down (NED) frame of (and centered at) `objIn` such that 0 degrees is North, 90 degrees is East, 180 degrees is South, and 270 degrees is West.

`el` — Elevation angles

scalar | vector

Elevation angles of target in the local AER system, returned as a scalar or vector. Elevations are measured with respect to a plane that is perpendicular to the normal of the surface of the earth. If `objIn` is on the surface of the Earth, then the plane is tangent to the Earth.

Values are specified in degrees in the closed interval [0 180]. The vector elements correspond to the time samples from the satellite scenario StartTime to StopTime properties, as specified by the SampleTime property. The elevation angle is defined in the NED frame of (and centered at) `objIn` such that 0 deg implies `target` is on the North East (NE) plane, 90 degrees implies `target` is directly above `objIn`, and -90 degrees implies `target` is directly below `objIn`.

`timeOut` — Time samples between start and stop time of scenario
scalar | vector | matrix | *N*-D array

Time samples between start and stop time of the scenario, returned as a scalar, vector, matrix, or *N*-D array. If `az` and `el` histories are returned

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `show` | `transmitter`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

show

Package: satelliteScenario

Show object in satellite scenario viewer

Syntax

```
show(item)
show(item,v)
```

Description

`show(item)` shows the item on all open Satellite Scenario Viewers.

`show(item,v)` shows the graphic on the Satellite Scenario Viewer specified by `v`.

Examples

Add Satellites to Scenario Using Keplerian Elements

Create a satellite scenario with a start time of 02-June-2020 8:23:00 AM UTC, and the stop time set to one day later. Set the simulation sample time to 60 seconds.

```
startTime = datetime(2020,6,02,8,23,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add two satellites to the scenario using their Keplerian elements.

```
semiMajorAxis = [10000000; 15000000];
eccentricity = [0.01; 0.02];
inclination = [0; 10];
rightAscensionOfAscendingNode = [0; 15];
argumentOfPeriapsis = [0; 30];
trueAnomaly = [0; 20];

sat = satellite(sc, semiMajorAxis, eccentricity, inclination, ...
    rightAscensionOfAscendingNode, argumentOfPeriapsis, trueAnomaly)
```

```
sat =
  1×2 Satellite array with properties:

    Name
    ID
    ConicalSensors
    Gimbals
    Transmitters
    Receivers
    Accesses
    GroundTrack
```

```
    Orbit
    OrbitPropagator
    MarkerColor
    MarkerSize
    ShowLabel
    LabelFontSize
    LabelFontColor
```

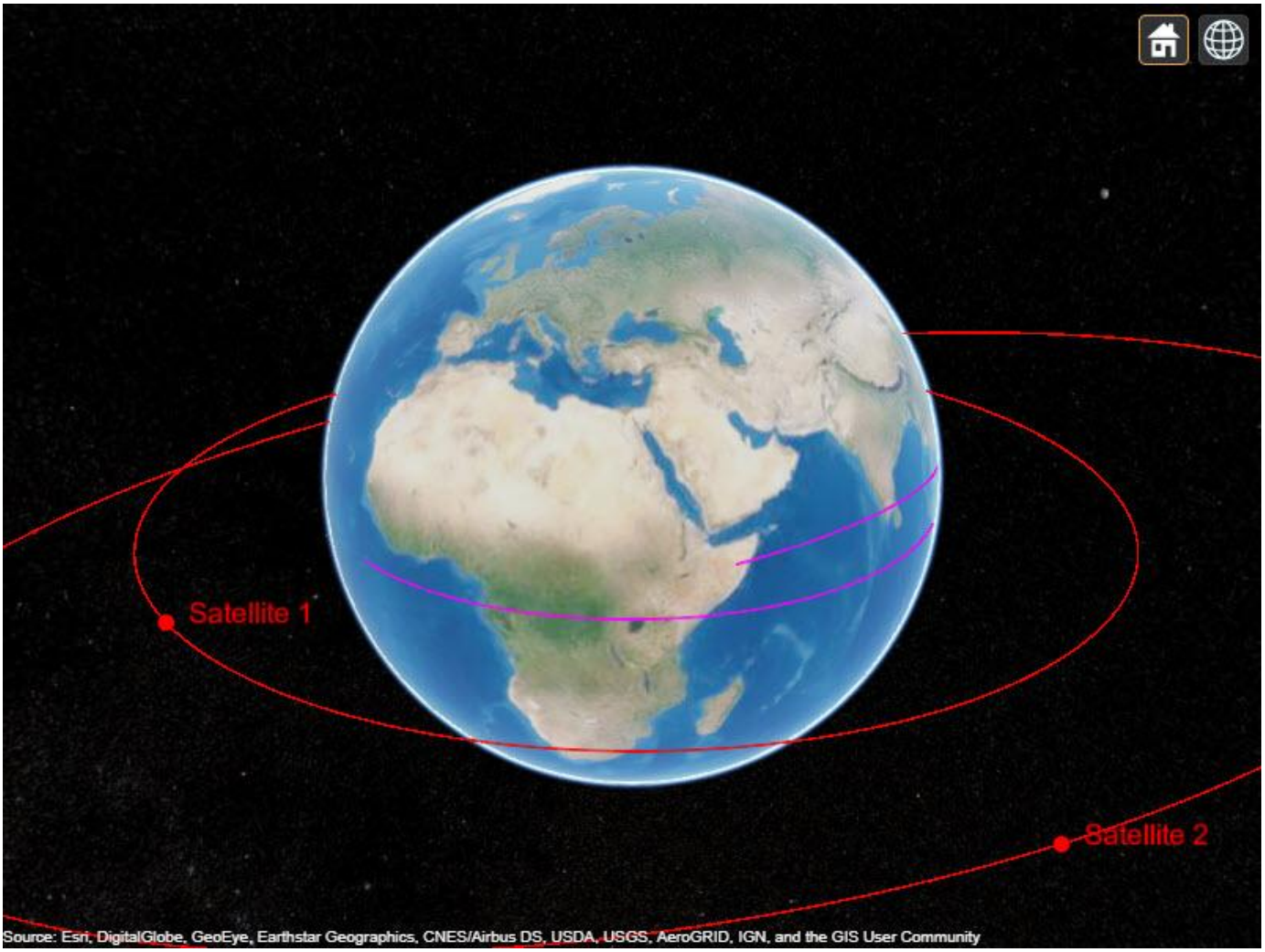
View the satellites in orbit and the ground tracks over one hour.

```
show(sat)
groundTrack(sat,'LeadTime',3600)
```

```
ans=1×2 object
  1×2 GroundTrack array with properties:

    LeadTime
    TrailTime
    LineWidth
    TrailLineColor
    LeadLineColor
    VisibilityMode
```

```
play(sc)
```



Input Arguments

`item` — Item

`Satellite` object | `GroundStation` object | `ConicalSensor` object | `GroundTrack` object | `FieldofView` object | `Access` object | `Link` object

`Satellite`, `GroundStation`, `ConicalSensors`, `GroundTrack`, `FieldOfView`, `Access` or `Link` object. These objects must belong to the same `satelliteScenario`, object.

Note If `item` is a satellite or a ground station, then the associated transmitters, receivers and gimbals are also displayed on the viewer.

`v` — Satellite scenario viewer

`satelliteScenarioViewer` object

Satellite scenario viewer, specified as a `satelliteScenarioViewer` object.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `transmitter`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

hide

Package: satelliteScenario

Hides satellite scenario entity from viewer

Syntax

```
hide(item)
hide(item,v)
```

Description

`hide(item)` hides `item` from all open satellite scenario viewers.

`hide(item,v)` hides the specified satellite scenario entity on the satellite scenario viewer specified by `v`.

Input Arguments

`item` — Item

`Satellite` object | `GroundStation` object | `ConicalSensor` object | `GroundTrack` object | `FieldofView` object | `Access` object | `Link` object

`Satellite`, `GroundStation`, `ConicalSensors`, `GroundTrack`, `FieldOfView`, `Access` or `Link` object. These objects must belong to the same `satelliteScenario`, object.

`v` — Satellite scenario viewer

`satelliteScenarioViewer` object

Satellite scenario viewer, specified as a `satelliteScenarioViewer`.

See Also

Objects
`satellite` | `satelliteScenarioViewer`

Functions
`access` | `groundStation` | `hideAll` | `play` | `satelliteScenario` | `show` | `showAll`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

ebno

Package: `satelliteScenario`

Eb/No at final node of link

Syntax

```
e = ebno(lnk)
e = ebno(lnk,timeIn)
[e,timeOut] = ebno( ___ )
```

Description

`e = ebno(lnk)` returns history of received energy per bit to noise power spectral density (Eb/No) values at the final node in a possibly multihop link.

`e = ebno(lnk,timeIn)` returns the received Eb/No values at the specified time.

`[e,timeOut] = ebno( ___ )` returns the received Eb/No values and the corresponding times in Universal Time Incorporated (UTC).

Input Arguments

`lnk` — Link analysis

`Link` object scalar

Link analysis object, specified as a `Link` object scalar.

`timeIn` — Time at which output is calculated

scalar

Time at which the output is calculated, specified as a scalar. If you do not specify a time zone, then the time zone is assumed to be UTC.

Output Arguments

`e` — Eb/No

scalar | vector

Eb/No, returned as a scalar or vector. If `timeIn` is not specified, `e` is a row vector.

`timeOut` — Time samples of output Eb/No

scalar | vector

Time samples of the output Eb/No, returned as a scalar or vector. If time history of Eb/No is returned, `timeOut` is a row vector.

See Also

Objects
`Link` | `satelliteScenario` | `satelliteScenarioViewer`

Functions
`hide` | `play` | `show`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

access

Package: `satelliteScenario`

Add access analysis objects to satellite scenario

Syntax

```
access(obj1,...,objN)
ac = access(obj1,...,objN)
ac = access( ___ ,'Viewer',Viewer)
```

Description

`access(obj1,...,objN)` adds `Access` objects defined by `obj1`, `obj2`, and so on.

`ac = access(obj1,...,objN)` returns a handle to the added access objects. The length of the vector corresponds to the number of `Access` objects added to the handle to the added access.

`ac = access( ___ ,'Viewer',Viewer)` sets the viewer in addition to any input argument combination from previous syntaxes. For example, `'Viewer',v1` picks the viewer `v1`.

Examples

Add Ground stations to Scenario and Visualize Access Intervals

Create satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020, 5, 1, 11, 36, 0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime, stopTime, sampleTime);
lat = [10];
lon = [-30];
gs = groundStation(sc, lat, lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
trueAnomaly = 0;
sat = satellite(sc, semiMajorAxis, eccentricity, inclination, ...
        rightAscensionOfAscendingNode, argumentOfPeriapsis, trueAnomaly);
```

Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

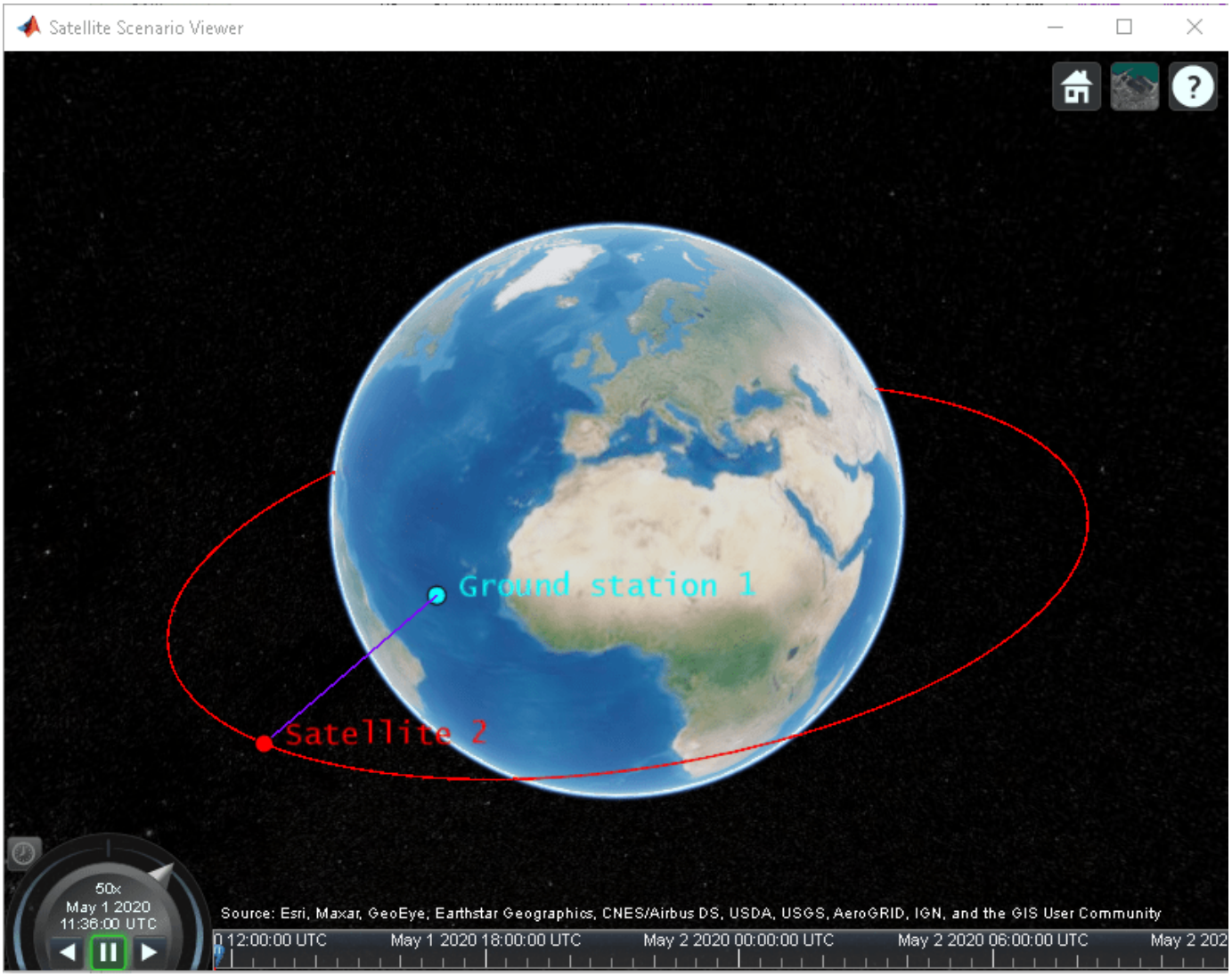
```
ac = access(sat, gs);
intvls = accessIntervals(ac)
```

```
intvls=8×8 table
       Source              Target          IntervalNumber        StartTime              EndTi
    _____________    ________________    _______________    ____________________    ____________

    "Satellite 2"    "Ground station 1"          1           01-May-2020 11:36:00    01-May-2020
    "Satellite 2"    "Ground station 1"          2           01-May-2020 14:20:00    01-May-2020
    "Satellite 2"    "Ground station 1"          3           01-May-2020 17:27:00    01-May-2020
    "Satellite 2"    "Ground station 1"          4           01-May-2020 20:34:00    01-May-2020
    "Satellite 2"    "Ground station 1"          5           01-May-2020 23:41:00    02-May-2020
    "Satellite 2"    "Ground station 1"          6           02-May-2020 02:50:00    02-May-2020
    "Satellite 2"    "Ground station 1"          7           02-May-2020 05:59:00    02-May-2020
    "Satellite 2"    "Ground station 1"          8           02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



Input Arguments

`obj1,...,objN` — Satellite, ground station, or conical sensor

`Satellite` object | `GroundStation` object | `ConicalSensor` object

`Satellite`, `GroundStation`, or `ConicalSensors` object. These objects must belong to the same `satelliteScenario` object. The function adds the access analysis object to the Accesses property of `obj1,...,objN`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Viewer',v1` picks the viewer `v1`.

`Viewer` — Satellite scenario viewer

(default) | row vector of all viewer objects | row vector of `satelliteScenarioViewer` objects

Satellite scenario viewer, specified as a row vector of `satelliteScenarioViewer` objects.

Data Types: `char` | `string`

Output Arguments

`ac` — Access analysis

`Access` object scalar

Access analysis between input objects, returned as an `Access` object scalar.

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `show` | `transmitter`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

groundStation

Add ground station to satellite scenario

Syntax

```
groundStation(scenario)
groundStation(scenario,lat,lon)
groundStation( ___ ,Name,Value)
gs = groundStation( ___ )
```

Description

`groundStation(scenario)` adds a default `GroundStation` object to the specified satellite scenario.

`groundStation(scenario,lat,lon)` sets the Latitude and Longitude properties of the ground station to `lat` and `lon`, respectively. `lat` and `lon` must be of the same length. This length specifies the number of ground stations that the function adds to the input scenario. Together, `lat` and `lon` indicate the locations of the ground stations.

`groundStation( ___ ,Name,Value)` sets options using one or more name-value arguments in addition to any input argument combination from previous syntaxes. For example, `'MinElevationAngle',10` specifies a minimum elevation angle of 10 degrees.

`gs = groundStation( ___ )` returns a vector of handles to the added ground stations. Specify any input argument combination from previous syntaxes.

Examples

Add Ground stations to Scenario and Visualize Access Intervals

Create satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020, 5, 1, 11, 36, 0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime, stopTime, sampleTime);
lat = [10];
lon = [-30];
gs = groundStation(sc, lat, lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
trueAnomaly = 0;
sat = satellite(sc, semiMajorAxis, eccentricity, inclination, ...
        rightAscensionOfAscendingNode, argumentOfPeriapsis, trueAnomaly);
```

Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

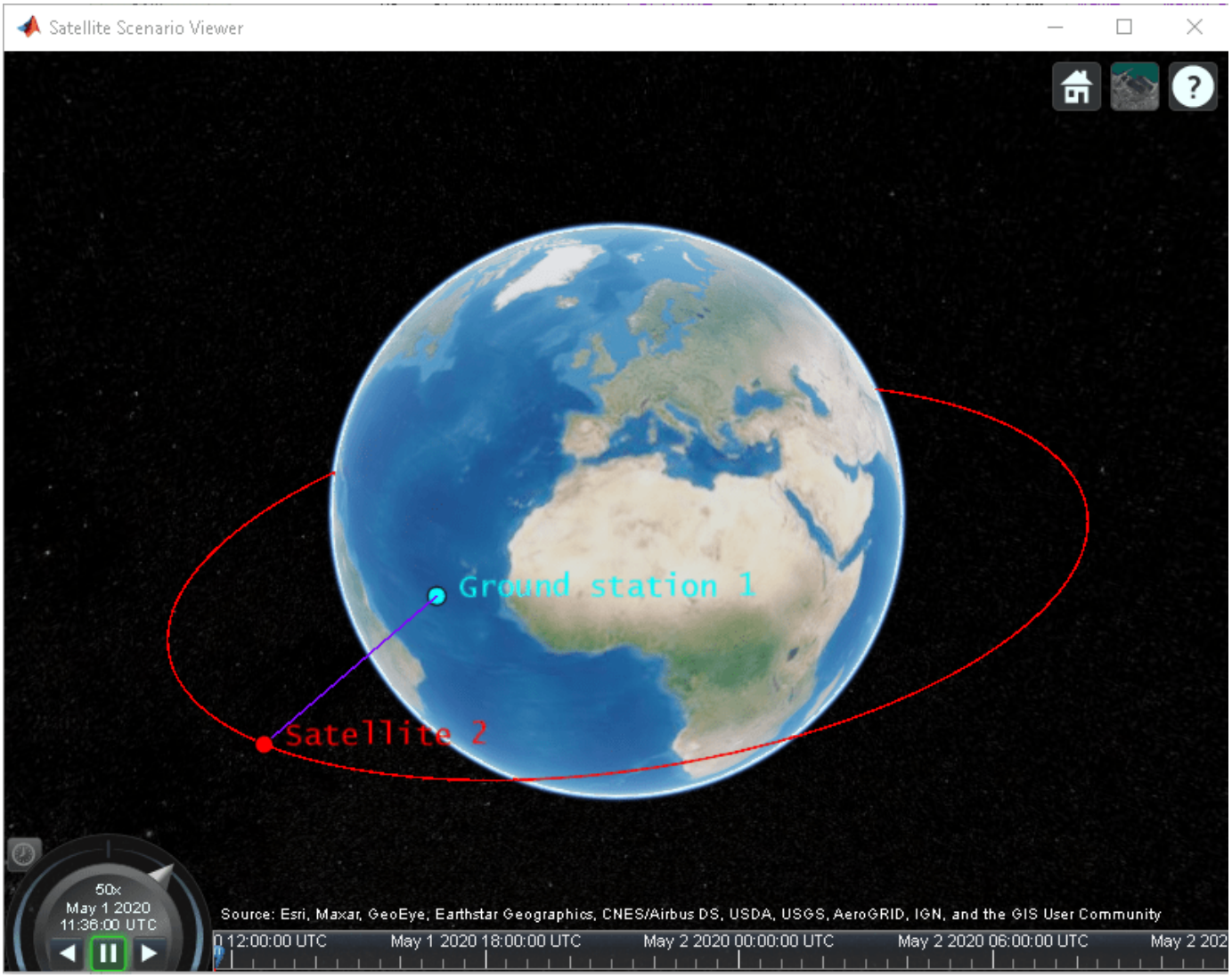
```
ac = access(sat, gs);
intvls = accessIntervals(ac)
```

```
intvls=8×8 table
       Source             Target          IntervalNumber         StartTime              EndTi
    ______________    ___________________    ______________    ____________________    _______

    "Satellite 2"    "Ground station 1"          1           01-May-2020 11:36:00    01-May-2020
    "Satellite 2"    "Ground station 1"          2           01-May-2020 14:20:00    01-May-2020
    "Satellite 2"    "Ground station 1"          3           01-May-2020 17:27:00    01-May-2020
    "Satellite 2"    "Ground station 1"          4           01-May-2020 20:34:00    01-May-2020
    "Satellite 2"    "Ground station 1"          5           01-May-2020 23:41:00    02-May-2020
    "Satellite 2"    "Ground station 1"          6           02-May-2020 02:50:00    02-May-2020
    "Satellite 2"    "Ground station 1"          7           02-May-2020 05:59:00    02-May-2020
    "Satellite 2"    "Ground station 1"          8           02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



Input Arguments

`scenario` — Satellite scenario

`satelliteScenario` object

Satellite scenario, specified as a `satelliteScenario` object.

`lat`, `lon` — Latitude and longitude

real-valued scalar | real-valued vector

Latitude and longitude of the ground station, specified as a real-valued scalar or real-valued vector.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MinElevationAngle',10` specifies a minimum elevation angle of 10 degrees.

`Viewer` — Satellite scenario viewer
(default) | row vector of all viewer objects | row vector of `satelliteScenarioViewer` objects

Satellite scenario viewer, specified as a row vector of `satelliteScenarioViewer` objects.

Data Types: `char` | `string`

`Name` — groundStation name
`"groundStation idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling groundStation. After you call groundStation, this property is read-only.

groundStation name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one groundStation is added, specify `Name` as a string scalar or a character vector.
- If multiple groundStations are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the groundStation added by the `groundStation` object function. If another groundStation of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`Latitude` — Geodetic latitude of ground stations
`42.3001` (default) | scalar | row vector

You can set this property only when calling groundStation. After you call groundStation, this property is read-only.

Geodetic latitude of ground stations, specified as a scalar. Values must be in the range [-90, 90].

- If you add only one ground station, specify Latitude as a scalar double.
- If you add multiple ground stations, specify Latitude as a vector double whose length is equal to the number of ground stations being added.

When latitude and longitude are specified as `lat, lon` inputs to groundStation, Latitude specified as a name-value argument takes precedence.

Data Types: `double`

`Longitude` — Geodetic longitude of ground stations
`-71.3504` (default) | scalar | row vector

You can set this property only when calling groundStation. After you call groundStation, this property is read-only.

Geodetic longitude of ground stations, specified as a scalar or a vector. Values must be in the range [-180, 180].

- If you add only one ground station, specify longitude as a scalar.
- If you add multiple ground stations, specify longitude as a vector whose length is equal to the number of ground stations being added.

When longitude and longitude are specified as `lat, lon` inputs to groundStation, longitude specified as a name-value argument takes precedence.

Data Types: `double`

`Altitude` — Altitude of ground station
`0` m (default) | scalar | vector

You can set this property only when calling groundStation. After you call groundStation, this property is read-only.

Altitude of ground stations, specified as a scalar or a vector.

- If you specify `Altitude` as a scalar, the value is assigned to each ground station in the groundStation.
- If you specify `Altitude` as a vector, the vector length must be equal to the number of ground stations in the groundStation.

When latitude and longitude are specified as `lat, lon` inputs to groundStation, Latitude specified as a name-value argument takes precedence.

Data Types: `double`

`MinElevationAngle` — Minimum elevation angle
`0` (default) | scalar | vector

Minimum elevation angle of a satellite for the satellite to be visible from the ground station, specified as a scalar or row vector. Values must be in the range [–90, 90]. For access and link closure to be possible, the elevation angle must be at least equal to the value specified in `MinElevationAngle`.

- If you specify `MinElevationAngle` as a scalar, the value is assigned to each ground station in the groundStation.
- If you specify `MinElevationAngle` as a vector, the vector length must be equal to the number of ground stations in the groundStation.

Data Types: `double`

Output Arguments

`gs` — Ground station in scenario
`GroundStation` object

Ground station in the scenario, returned as a `GroundStation` object belonging to the satellite scenario specified by the input `scenario`.

You can modify the `GroundStation` object by changing its property values. The name-value arguments used when calling this function correspond to property names.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `hide` | `play` | `receiver` | `satellite` | `show` | `transmitter`

Topics
“Multi-Hop Satellite Communications Link Between Two Ground Stations”
“Satellite Constellation Access to a Ground Station”
“Comparison of Orbit Propagators”
“Modeling Satellite Constellations using Ephemeris Data”
“Estimate GNSS Receiver Position with Simulated Satellite Constellations”
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

transmitter

Package: satelliteScenario

Add transmitter to satellite scenario

Syntax

```
transmitter(parent)
transmitter(parent,Name,Value)
tx = transmitter( ___ )
```

Description

`transmitter(parent)` adds a default `Transmitter` object to the `parent` which can be a `Satellite`, `GroundStation` or `Gimbal`.

`transmitter(parent,Name,Value)` specifies options using one or more name-value arguments. For example, `'MountingAngle',[20; 35; 10]` sets the yaw, pitch, and roll angles of the transmitter to 20, 35, and 10 degrees, respectively.

`tx = transmitter( ___ )` returns a handle to the added transmitter. Specify any input argument combination from previous syntaxes.

Input Arguments

`parent` — Element of scenario to which transmitter is added

`Satellite` object | `GroundStation` object | `Gimbal` object

Element of scenario to which the transmitter is added, specified as a `Satellite`, `GroundStation`, or `Gimbal` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MountingAngle',[20; 35; 10]` sets the yaw, pitch, and roll angles of the transmitter to 20, 35, and 10 degrees, respectively.

`Name` — transmitter name

`"transmitter idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling transmitter. After you call transmitter, this property is read-only.

transmitter name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one transmitter is added, specify `Name` as a string scalar or a character vector.
- If multiple transmitters are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the transmitter added by the `transmitter` object function. If another transmitter of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting location with respect to the parent object, specified as a three-element row vector of positive numbers in meters. The position vector is specified in the body frame of the input `parent`.

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting orientation with respect to parent object, specified as a three-element row vector of positive numbers in degrees. The elements of the vector correspond to yaw, pitch, and roll in that order. Yaw, pitch, and roll are positive rotations about the parent's *z* - axis, intermediate *y* - axis and intermediate *x* - axis of the parent.

Example: `[0; 30; 60]`

`Antenna` — Antenna object associated with transmitter
`gaussianAntenna` object

Antenna object associated with the transmitter, specified as an antenna object. The default gaussian antenna has a dish diameter of 1 m and an aperture efficiency of 0.65.

`SystemLoss` — Total loss in transmitter
`5` (default) | positive scalar

Total loss in the transmitter, specified as a real positive scalar. Units are in dB.

`Frequency` — Transmitter frequency
`14e9` (default) | positive scalar

Transmitter frequency, specified as a positive scalar. Units are in Hz.

`BitRate` — Bit rate of transmitter
`10` (default) | real positive scalar

Bit rate of the transmitter, specified as a real positive scalar. Units are in Mbps.

`Power` — Power of high power amplifier
`12` (default) | real positive scalar

Power of the high power amplifier, specified as a real positive scalar. Units are in dbW.

Output Arguments

`tx` — Transmitter
`Transmitter` object

Transmitter attached to `parent`, returned as a `Transmitter` object.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `groundStation` | `hide` | `link` | `play` | `receiver` | `show`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

receiver

Package: `satelliteScenario`

Add receiver to satellite scenario

Syntax

```
receiver(parent)
receiver(parent,Name,Value)
rx = receiver( ___ )
```

Description

`receiver(parent)` adds a default `Receiver` object to the `parent` which can be a `Satellite`, `GroundStation` or `Gimbal`.

`receiver(parent,Name,Value)` specifies options using one or more name-value arguments. For example, `'MountingAngle',[20; 35; 10]` sets the yaw, pitch, and roll angles of the transmitter to 20, 35, and 10 degrees, respectively.

`rx = receiver( ___ )` returns a handle to the added receiver. Specify any input argument combination from previous syntaxes.

Input Arguments

`parent` — Element of scenario to which receiver is added

`Satellite` object | `GroundStation` object | `Gimbal` object

Element of scenario to which the receiver is added, specified as a `Satellite`, `GroundStation`, or `Gimbal` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MountingAngle',[20; 35; 10]` sets the yaw, pitch, and roll angles of the receiver to 20, 35, and 10 degrees, respectively.

`Name` — receiver name

`"receiver idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling receiver. After you call receiver, this property is read-only.

receiver name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one receiver is added, specify `Name` as a string scalar or a character vector.

- If multiple receivers are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the receiver added by the `receiver` object function. If another receiver of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting location with respect to the parent object, specified as a three-element row vector of positive numbers in meters. The position vector is specified in the body frame of the input `parent`.

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting orientation with respect to parent object, specified as a three-element row vector of positive numbers in degrees. The elements of the vector correspond to yaw, pitch, and roll in that order. Yaw, pitch, and roll are positive rotations about the parent's z - axis, intermediate y - axis and intermediate x - axis of the parent.

Example: `[0; 30; 60]`

`Antenna` — Antenna object associated with receiver
`gaussianAntenna` object

Antenna object associated with the receiver, specified as an antenna object. The default gaussian antenna has a dish diameter of 1 m and an aperture efficiency of 0.65.

`SystemLoss` — Total loss in receiver
`5` (default) | positive scalar

Total loss in the receiver, specified as a real positive scalar. Units are in dB.

`GainToNoiseTemperatureRatio` — Gain to noise temperature ratio
`3` (default) | scalar

Gain to noise temperature ratio of the antenna, specified as the comma-separated pair consisting of `'GainToNoiseTemperatureRatio'` and a scalar. Units are in dB/K.

`RequiredEbNo` — Lowest Eb/No necessary for link closure
`10` (default) | positive scalar

Lowest energy per bit to noise power spectral density ratio (Eb/No) necessary for link closure, specified as the comma-separated pair consisting of `'RequiredEbNo'` and a positive scalar. Units are in dB.

Output Arguments

`rx` — Receiver
`Receiver` object

Receiver attached to `parent`, returned as a `Receiver` object.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `groundStation` | `hide` | `link` | `play` | `show` | `transmitter`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

gimbal

Add gimbal to satellite or ground station

Syntax

```
gimbal(parent)
gimbal(parent,Name,Value)
gimbal( ___ )
```

Description

`gimbal(parent)` adds a default `Gimbal` object to `parent`, which can be a satellite, ground station, or gimbal.. A gimbal can dynamically change orientation independent of the parent. Transmitters, receivers, and conical sensors can be mounted on the gimbals.

`gimbal(parent,Name,Value)` specifies options using one or more name-value arguments.

`gim = gimbal( ___ )` returns a handle to the added gimbal. Specify any input argument combination from previous syntaxes.

Input Arguments

`parent` — Element of scenario to which gimbal is added

`Satellite` object | `GroundStation` object | `Gimbal` object

Element of scenario to which the gimbal is added, specified as a `Satellite`, `GroundStation`, or `Gimbal` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MountingAngle',[20; 35; 10]` sets the yaw, pitch, and roll angles of gimbal to 20, 35, and 10 degrees, respectively.

`Name` — gimbal name

`"gimbal idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling gimbal. After you call gimbal, this property is read-only.

gimbal name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one gimbal is added, specify `Name` as a string scalar or a character vector.
- If multiple gimbals are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the gimbal added by the `gimbal` object function. If another gimbal of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting location with respect to the parent object, specified as a three-element row vector of positive numbers in meters. The position vector is specified in the body frame of the input `parent`.

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting orientation with respect to parent object, specified as a three-element row vector of positive numbers in degrees. The elements of the vector correspond to yaw, pitch, and roll in that order. Yaw, pitch, and roll are positive rotations about the parent's z - axis, intermediate y - axis and intermediate x - axis of the parent.

Example: `[0; 30; 60]`

Output Arguments

`gim` — Gimbal
`Gimbal` object

Gimbal attached to `parent`, returned as a `Gimbal` object.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `conicalSensor` | `groundStation` | `hide` | `play` | `satellite` | `show`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

fieldOfView

Package: `satelliteScenario`

Visualize field of view of conical sensor

Syntax

```
fieldOfView(sensor)
fieldOfView(sensor,Name,Value)
fov = fieldOfView( ___ )
```

Description

`fieldOfView(sensor)` adds a `FieldOfView` object to the specified conical sensor, and draws contours on the Earth. Each contour represents the field of view of a conical sensor in `sensor` based on the current state of the scenario.

Locations inside the contour are inside the field of view. If no viewer is open, a new viewer is launched, and the field of view contours are shown in the open viewer. If a viewer is already open, the field of view contours are added to it. The contours are the lines of intersection of the surface of the earth and the field of view cone. The half angle of the field of view cone is equal to the MaxViewAngle property of the conical sensor, and the axis of the cone is the *z*-axis (or boresight) of the conical sensor. The vertex of the cone is located at the position of the conical sensor. The cone becomes wider along the positive body *z*-axis of the conical sensor.

`fieldOfView(sensor,Name,Value)` specifies options by using one or more name-value arguments.

`fov = fieldOfView( ___ )` returns a vector of handles to the added field of view graphic objects. Specify any input combination from previous syntaxes.

Input Arguments

`sensor` — Conical sensor

`ConicalSensor` object

Conical sensor, specified as a `ConicalSensor` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'LineWidth',2.5` sets the line width of the field of view to 2.5 pixels.

`LineWidth` — Visual width of field of view contour

`1` (default) | scalar in the range (0 10]

Visual width of the field of view contour in pixels, specified as a scalar in the range (0 10].

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LineColor` — Color of field of view contour
`[0 1 0]` (default) | RGB triplet | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of field of view contour, specified as an RGB triplet, hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	
`[0.8500 0.3250 0.0980]`	`'#D95319'`	
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

Output Arguments

`fov` — Field of view of conical sensor

row vector of `FieldOfView` objects

Field of view of conical sensor, returned as a row vector of `FieldOfView` objects.

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`access` | `conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `show` | `transmitter`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

link

Package: satelliteScenario

Add link analysis objects to transmitter

Syntax

```
link(obj1,...,objN)
lnk = link( ___ )
```

Description

`link(obj1,...,objN)` adds `Link` objects defined by `obj1`, `obj2`, and so on..

`lnk = link( ___ )` returns a handle to the added `Link` object.

Input Arguments

`obj1,...,objN` — Satellite, ground station, or conical sensor

`Transmitter` object | `Receiver` object

`Transmitter` or `Receiver` object, specified as separate arguments where the `obj1` must be a `Transmitter` object and any following arguments can be `Transmitter` or `Receiver` objects. These arguments specify the Sequence of the link. These objects must belong to the same `satelliteScenario` object. The function adds the link analysis object to the Link property of `obj1`.

Output Arguments

`lnk` — Link analysis

`Link` object scalar

Link analysis between input objects, returned as a row vector of `Link` objects.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`groundStation` | `hide` | `play` | `receiver` | `show` | `transmitter`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

gaussianAntenna

Package: satelliteScenario

Add Gaussian antennas

Syntax

```
gaussianAntenna(trx)
gaussianAntenna(trx,Name,Value)
ant = gaussianAntenna( ___ )
```

Description

`gaussianAntenna(trx)` adds `GaussianAntenna` object to the specified transmitter or receiver. The gaussian antenna is assigned to the Antenna property by overwriting it.

`gaussianAntenna(trx,Name,Value)` adds an antenna and specifies options using one or more name-value arguments. Enclose each property name in quotes. For example, `'DishDiameter',1.7` sets the dish diameter of the antenna to 1.7 meters upon creation.

`ant = gaussianAntenna( ___ )` adds an antenna and returns a handle to the added `GaussianAntenna` object. You can add only one `GaussianAntenna` to a given `Transmitter` or `Receiver`.

Input Arguments

`trx` — Transmitter or receiver

`Transmitter` object | `Receiver` object

Transmitter or receiver to which the gaussian antenna is added, specified as a `Transmitter` or `Receiver` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'DishDiameter',1.7` sets the dish diameter of the antenna to 1.7 meters upon creation.

`DishDiameter` — Diameter of the antenna dish

`1` (default) | positive scalar

You can set this property only when calling `gaussianAntenna`. After you call gaussianAntenna, this property is read-only.

Diameter of the Gaussian antenna dish, specified as a real positive scalar. Units are in meters.

`ApertureEfficiency` — Aperture efficiency of Gaussian antenna

`0.65` (default) | scalar in the range (0,1]

You can set this property only when calling gaussianAntenna. After you call gaussianAntenna, this property is read-only.

Aperture efficiency of the Gaussian antenna, specified as a scalar in the range (0,1].

Output Arguments

ant — Gaussian antenna

`GaussianAntenna` object scalar

Gaussian antenna added to the specified transmitter or receiver, returned as a `GaussianAntenna` object scalar.

See Also

Objects

`satelliteScenario`

Functions

`access` | `groundStation` | `hide` | `play` | `receiver` | `satellite` | `show` | `transmitter`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

groundTrack

Package: satelliteScenario

Add ground track object to satellite in scenario

Syntax

```
groundTrack(sat)
groundTrack( ___ ,Name,Value)
```

Description

`groundTrack(sat)` adds ground track visualization for each satellite in `sat` based on their current positions. The ground track begins at the scenario StartTime, and ends at the StopTime. The spacing between samples that make up the ground track visualization is determined by the scenario `SampleTime`. If no viewer is open, a new viewer is launched, and the ground track is displayed. If a viewer is already open, the ground track is added to that viewer. By default, ground tracks will be displayed in 2-D.

`groundTrack( ___ ,Name,Value)` adds a `groundTrack` object by using one or more name-value pairs. Enclose each property name in quotes.

Input Arguments

`sat` — Satellite

row vector of `Satellite` objects

Satellite, specified as a row vector of `Satellite` objects.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'LeadTime',3600` sets the lead time of the ground track to 3600 seconds upon creation.

`Viewer` — Viewer on which ground track is visualized

`satelliteScenarioViewer` object

Viewer on which ground track is visualized, specified as a `satelliteScenarioViewer` object.

`LeadTime` — Period of future ground track to be visualized

`StartTime` to `StopTime` (default) | real positive scalar

Period of future ground track to be visualized in `Viewer`, specified as a comma-separated pair consisting of `'LeadTime'` and a real positive scalar in seconds.

`TrailTime` — Period of ground track history to be visualized

`StartTime` to `StopTime` (default) | real positive scalar

Period of ground track history to be visualized in `Viewer`, specified as a comma-separated pair consisting of `'TrailTime'` and a real positive scalar in seconds.

`LineWidth` — Visual width of ground track
1 (default) | scalar

Visual width of ground track in pixels, specified as a comma-separated pair consisting of `'LineWidth'` and a scalar in the range (0,10).

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LeadTime` — Period of ground track to be visualized
`StartTime` to `StopTime` (default) | positive scalar

Period of the ground track to be visualized in the satellite scenario viewer, specified as a comma-separated pair consisting of `'LeadTime'` and a real positive scalar in seconds.

`TrailTime` — Period of ground track history to be visualized
`StartTime` to `StopTime` (default) | positive scalar

Period of the ground track history to be visualized in `Viewer`, specified as a comma-separated pair consisting of `'TrailTime'` and a real positive scalar in seconds.

`LineWidth` — Visual width of ground track
1 (default) | scalar in the range (0 10]

Visual width of the ground track in pixels, specified as a comma-separated pair consisting of `'LineWidth'` and a scalar in the range (0 10].

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LeadLineColor` — Color of future ground track line
`[1 0 1]` (default) | RGB triplet | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the future ground track line, specified as a comma-separated pair consisting of `'LeadLineColor'` and an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'red'	'r'	[1 0 0]	'#FF0000'	
'green'	'g'	[0 1 0]	'#00FF00'	
'blue'	'b'	[0 0 1]	'#0000FF'	
'cyan'	'c'	[0 1 1]	'#00FFFF'	
'magenta'	'm'	[1 0 1]	'#FF00FF'	
'yellow'	'y'	[1 1 0]	'#FFFF00'	
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	
'none'	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

Example: 'blue'

Example: [0 0 1]

Example: '#0000FF'

TrailLineColor — Color of ground track line history

[1 0.5 0] (default) | RGB triplet | RGB triplet | string scalar of color name | character vector of color name

Color of the ground track line history, specified as a comma-separated pair consisting of 'TrailLineColor' and an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from 0 to F. The values are not case sensitive. Thus, the color codes '#FF8800', '#ff8800', '#F80', and '#f80' are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	
`[0.8500 0.3250 0.0980]`	`'#D95319'`	
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `groundStation` | `hide` | `play` | `satellite` | `show`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

Objects

ccsdsTCConfig

Create CCSDS TC configuration object

Description

The `ccsdsTCConfig` object creates a configuration object for Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) using default and specified values. `ccsdsTCConfig` object is configurable by using applicable “Properties” on page 3-2.

Creation

Syntax

```
cfg = ccsdsTCConfig
cfg = ccsdsTCConfig(Name,Value)
```

Description

`cfg = ccsdsTCConfig` creates a CCSDS TC configuration object using default properties.

`cfg = ccsdsTCConfig(Name,Value)` sets “Properties” on page 3-2 using one or more name-value pairs. Enclose each property name in quotes. For example, `ccsdsTCConfig('DataFormat','CLTU','Modulation','BPSK')` configures the CSSDS TC configuration object with a communications link transmission unit data format and binary phase shift keying (BPSK) modulation scheme.

Properties

`DataFormat` — Data formats used by PLOPs
`"CLTU"` (default) | `"acquisition sequence"` | `"idle sequence"`

Data formats used by physical layer operation procedures (PLOPs), specified as one of these options.

- `"CLTU"` — Communications link transmission unit (CLTU)
- `"acquisition sequence"`
- `"idle sequence"`

Data Types: `char` | `string`

`ChannelCoding` — Forward error correction coding
`"BCH"` (default) | `"LDPC"`

Forward error correction coding, specified as one of these options.

- `"BCH"` — Bose Chaudhuri Hocquenghem (BCH)
- `"LDPC"` — Low-density parity–check (LDPC)

Dependencies

To enable this property, set the `DataFormat` property to `"CLTU"`.

Data Types: `char` | `string`

`LDPCCodewordLength` — LDPC codeword length

`128` (default) | `512`

LDPC codeword length, specified as `128` or `512`.

Dependencies

To enable this property, set the `ChannelCoding` property to `"LDPC"`.

Data Types: `double`

`HasRandomizer` — Flag to indicate randomization

`1` or `true` (default) | `0` or `false`

Flag to indicate randomization on the bits in CLTU and on the fill data added prior to randomization, specified as a logical value of `1` (`true`) or `0` (`false`). To indicate the presence of a randomizer in the waveform, set this value to `1` (`true`).

Dependencies

To enable this property, set the `ChannelCoding` property to `"BCH"`.

Data Types: `logical`

`HasTailSequence` — Flag to indicate tail sequence in CLTU

`1` or `true` (default) | `0` or `false`

Flag to indicate the tail sequence in CLTU, specified as a logical value of `1` (`true`) or `0` (`false`). To indicate the presence of the tail sequence to delimit the end of a CLTU, set this value to `1` (`true`).

Dependencies

To enable this property, set the `ChannelCoding` property to `"LDPC"` and the `LDPCCodewordLength` property to `128`.

Data Types: `logical`

`Modulation` — Modulation scheme

`"PCM/PSK/PM"` (default) | `"PCM/PM/biphase-L"` | `"BPSK"`

Modulation scheme used to generate the CCSDS TC waveform, in the form of baseband in-phase quadrature (IQ) samples, specified as one of these options.

- `"PCM/PSK/PM"` — The line coded signal as per the pulse code modulation (PCM) format is phase shift keying (PSK) modulated on a sine wave subcarrier and then phase modulated (PM) on a residual carrier.
- `"PCM/PM/biphase-L"` — The biphase-L (Manchester) encoded data is phase modulated on a residual carrier.
- `"BPSK"` — Suppressed carrier modulation by using non-return-to-zero (NRZ) data on the carrier.

For more details on these modulation schemes, see [3].

Data Types: `char` | `string`

`PCMFormat` — PCM format

`"NRZ-L"` (default) | `"NRZ-M"`

Pulse code modulation (PCM) format, specified as one of these options. This property specifies the PCM coding in the CCSDS TC waveform.

- `"NRZ-L"` — NRZ-level
- `"NRZ-M"` — NRZ-mark

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `char` | `string`

`ModulationIndex` — Modulation index in residual carrier phase modulation

`0.4` (default) | scalar in the range [0.2, 2]

Modulation index in the residual carrier phase modulation, specified as a scalar in the range [0.2, 2]. Units are in radians.

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"` or `"PCM/PM/biphase-L"`.

Data Types: `double`

`SubcarrierFrequency` — Sine wave subcarrier frequency

`16000` (default) | `8000`

Sine wave subcarrier frequency in Hertz, specified as `16000` or `8000`. The subcarrier waveform is used to PSK-modulate the NRZ data on the residual RF carrier.

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `double`

`SymbolRate` — Symbol rate

`4000` (default) | `2000` | `1000` | `500` | `250` | `125` | `62.5` | `31.25` | `15.625` | `7.8125`

Symbol rate in coded symbols per second, specified as one of these options.

- `4000`
- `2000`
- `1000`
- `500`
- `250`
- `125`
- `62.5`
- `31.25`

- 15.625
- 7.8125

Note If you set `SymbolRate` to `4000` coded symbols per second, you must set the `SubcarrierFrequency` property to `16000`.

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `double`

`SamplesPerSymbol` — Number of samples per symbol

`10` (default) | positive integer

Number of samples per symbol, specified as a positive integer.

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"` or `"PCM/PM/biphase-L"`.

Data Types: `double`

`SubcarrierWaveform` — Waveform used to PSK-modulate NRZ data

`"sine"`

This property is read-only.

Waveform used to PSK-modulate the NRZ data, returned as `"sine"`. CCSDS TC supports only sine-wave subcarriers.

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `char` | `string`

Object Functions

Specific to This Object

ccsdsTCWaveform Generate CCSDS TC waveform

Examples

Create CCSDS TC Object

Create a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) configuration object. Specify the properties of the object.

```
cfg = ccsdsTCConfig;
cfg.ChannelCoding = "LDPC";
cfg.HasTailSequence = false;
cfg.PCMFormat = "NRZ-M";
```

Display the properties of the CCSDS TC object.

```
disp(cfg)
```

```
  ccsdsTCConfig with properties:

            DataFormat: "CLTU"
          ChannelCoding: "LDPC"
     LDPCCodewordLength: 128
        HasTailSequence: 0
             Modulation: "PCM/PSK/PM"
              PCMFormat: "NRZ-M"
        ModulationIndex: 0.4000
    SubcarrierFrequency: 16000
             SymbolRate: 4000
       SamplesPerSymbol: 10

   Read-only properties:
     SubcarrierWaveform: "sine"
```

Create CCSDS TC Waveform for Multiple CLTUs

Create a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) time-domain waveform for multiple communications link transmission units (CLTUs).

Create a default CCSDS TC configuration object.

```
cfg = ccsdsTCConfig;
disp(cfg)
```

```
  ccsdsTCConfig with properties:

            DataFormat: "CLTU"
          ChannelCoding: "BCH"
          HasRandomizer: 1
             Modulation: "PCM/PSK/PM"
              PCMFormat: "NRZ-L"
        ModulationIndex: 0.4000
    SubcarrierFrequency: 16000
             SymbolRate: 4000
       SamplesPerSymbol: 10

   Read-only properties:
     SubcarrierWaveform: "sine"
```

Specify the number of CLTUs and the transfer frame length.

```
numCLTUs = 10;
transferFramesLength = 8; % Number of octets in each transfer frame
```

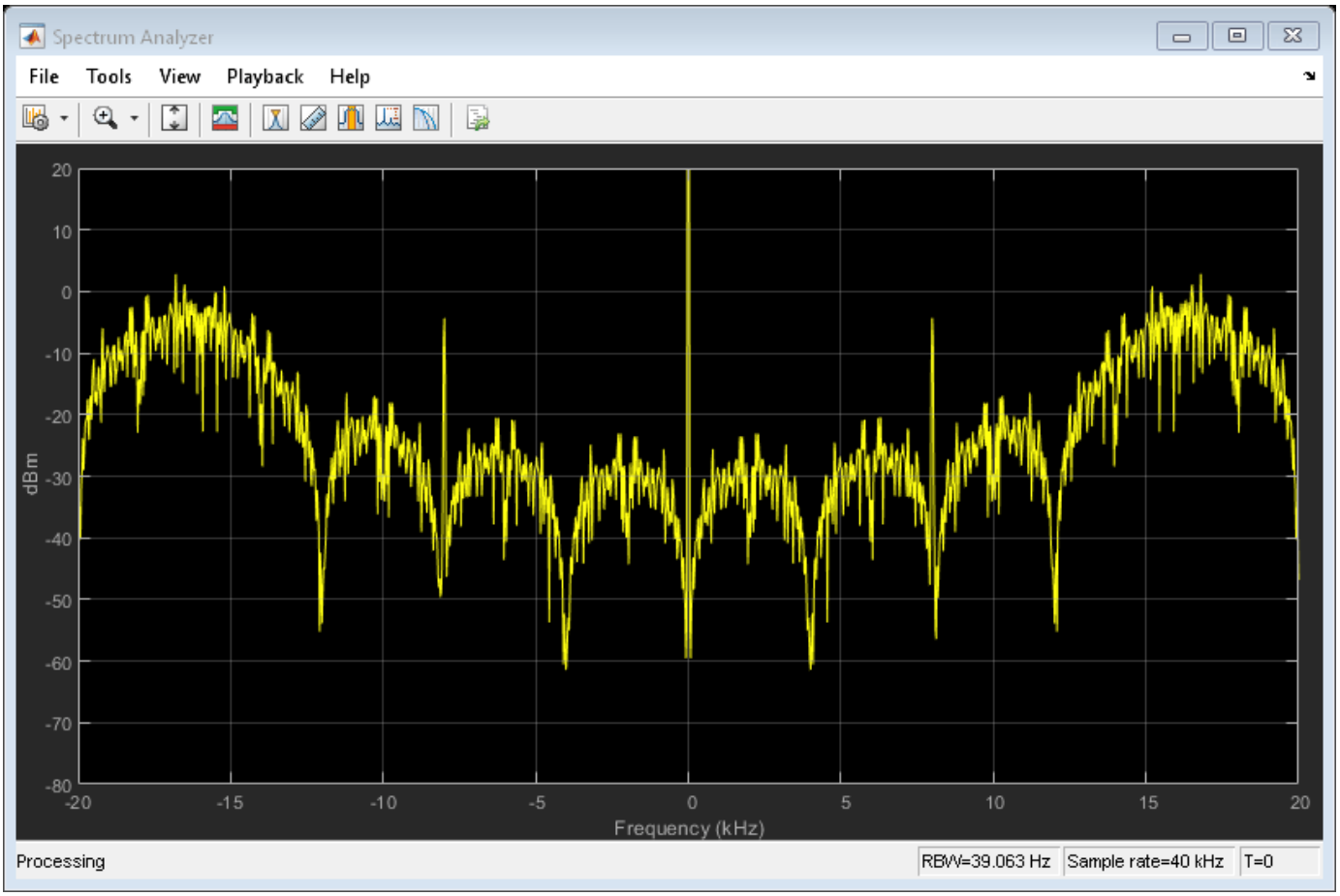
Generate the CCSDS TC time-domain waveform for the transfer frames.

```
c = cell(1,numCLTUs); % Cell array to store the generated waveform for all CLTUs
for k=1:numCLTUs
    bits = randi([0 1],8*transferFramesLength,1); % Bits in the TC transfer frame
    waveform = ccsdsTCWaveform(bits,cfg);
```

```
    c{1,k} = waveform; % Waveform for each CLTU
end
```

Create a `dsp.SpectrumAnalyzer` System object to display the frequency spectrum of the generated CCSDS TC time-domain waveform from the last CLTU.

```
scope = dsp.SpectrumAnalyzer;
scope.SampleRate = cfg.SamplesPerSymbol*cfg.SymbolRate;
scope(waveform)  % Last CLTU spectrum display
```



References

[1] CCSDS 231.0-B-3. Blue Book. Issue 3. "TC Synchronization and Channel Coding." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, September 2017.

[2] CCSDS 401.0-B-29. Blue Book. Issue 29. "Radio Frequency and Modulation Systems - Part 1". *Earth Stations and Spacecraft*. Washington, D.C.: CCSDS, September 2019.

[3] Nguyen, T.M., W.L. Martin, and Hen-Geul Yeh. "Required Bandwidth, Unwanted Emission, and Data Power Efficiency for Residual and Suppressed Carrier Systems - a Comparative Study." *IEEE transactions on electromagnetic compatibility* 37, no. 1 (February 1995): 34-50. https://doi.org/10.1109/15.350238.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Properties `LDPCCodewordLength` and `ChannelCoding` must be provided as compile-time constant inputs in code generation. Use `coder.Constant` (MATLAB Coder) object to convert the input variable to a constant during code generation.

See Also

Functions

`ccsdsTCIdealReceiver` | `ccsdsTCWaveform`

Introduced in R2021a

p618SiteDiversityConfig

Create P.618 site diversity configuration object

Description

The `p618SiteDiversityConfig` object sets P.618 site diversity configuration parameters required for the calculation of outage probability due to rain attenuation, as defined in the ITU-R P.618 recommendation [1].

Creation

Syntax

```
cfgSD = p618SiteDiversityConfig
cfgSD = p618SiteDiversityConfig(Name,Value)
```

Description

`cfgSD = p618SiteDiversityConfig` creates a P.618 site diversity configuration object with default property values.

`cfgSD = p618SiteDiversityConfig(Name,Value)` specifies “Properties” on page 3-9 using one or more name-value pair arguments. Enclose each property name in quotes. For example, p618SiteDiversityConfig(`'Frequency',14.25e9,'ElevationAngle',[52.4099 52.4852]`) configures a P.618 site diversity configuration object with a 14.25 GHz signal frequency and an elevation angle for two sites as [52.4099 52.4852].

Properties

`Frequency` — Signal frequency

`14.25e9` (default) | scalar in the range [1e9, 55e9]

Signal frequency in Hz, specified as a scalar in the range [1e9, 55e9].

Data Types: `double` | `single`

`ElevationAngle` — Elevation angle of two sites

`[52.4099 52.4852]` (default) | two-element vector of values in the range [0, 90]

Elevation angle of the two sites in degrees, specified as a two-element vector of values in the range [0, 90].

Data Types: `double` | `single`

`Latitude` — Latitude of two sites

`[25.768 25.463]` (default) | two-element vector of values in the range [-90, 90]

Latitude of the two sites in degrees, specified as a two-element vector of values in the range [-90, 90]. A positive value corresponds to a North latitude, and a negative value corresponds to a South latitude.

Data Types: `double` | `single`

`Longitude` — Longitude of two sites
`[-80.205 -80.486]` (default) | two-element vector of values in the range [-180, 180]

Longitude of the two sites in degrees, specified as a two-element vector of values in the range [-180, 180]. A positive value corresponds to East longitude, and a negative value corresponds to West longitude.

Data Types: `double` | `single`

`PolarizationTiltAngle` — Polarization tilt angle for two sites
`[0 0]` (default) | two-element vector of values in the range [-90, 90]

Polarization tilt angle for the two sites in degrees, specified as a two-element vector of values in the range [-90, 90].

Data Types: `double` | `single`

`SiteDistance` — Separation between two sites
`44.0256` (default) | positive scalar

Separation between the two sites in km, specified as a positive scalar.

Data Types: `double` | `single`

`AttenuationThreshold` — Attenuation threshold on two links
`[9 3]` (default) | two-element vector

Attenuation threshold on the two links in dB, specified as a two-element vector. The attenuation threshold on an earth space link is the maximum allowed attenuation on the path. Any attenuation value above this property value is considered an outage in the link.

Data Types: `double` | `single`

Object Functions

Specific to This Object

p618SiteDiversityOutage Calculate outage probability due to rain attenuation with site diversity

Examples

Create P.618 Site Diversity Configuration Object

Create a default P.618 site diversity configuration object.

```
cfg = p618SiteDiversityConfig;
```

Specify the polarization tilt angles for two sites as [-90 90] degrees, separation between the two sites as 50 km, and attenuation threshold on the two links as [9 9] dB.

```
cfg.PolarizationTiltAngle = [-90 90];
cfg.SiteDistance = 50;
cfg.AttenuationThreshold = [9 9];
```

Set the direction of each earth station.

```
cfg.Latitude = [30 60];    % North direction
cfg.Longitude = [120 150]; % East direction
```

Display the properties of the configuration object.

```
disp(cfg);
```

```
  p618SiteDiversityConfig with properties:

               Frequency: 1.4500e+10
          ElevationAngle: [52.4099 52.4852]
                Latitude: [30 60]
               Longitude: [120 150]
    PolarizationTiltAngle: [-90 90]
             SiteDistance: 50
     AttenuationThreshold: [9 9]
```

Calculate Outage Probability due to Rain Attenuation with Site Diversity

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and untar the MAT-files.

```
if ~exist('ITURDigitalMaps.tar.gz','file')
    url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
    websave('ITURDigitalMaps.tar.gz',url);
    untar('ITURDigitalMaps.tar.gz');
end
```

Create a P.618 site diversity configuration object with a signal frequency of 25 GHz.

```
cfgsd = p618SiteDiversityConfig;
cfgsd.Frequency = 25e9;
```

Specify the polarization tilt angles for two sites as [-90 90] degrees, separation between the two sites as 50 km, and attenuation threshold on the two links as [9 9] dB.

```
cfgsd.PolarizationTiltAngle = [-90 90];
cfgsd.SiteDistance = 50;
cfgsd.AttenuationThreshold = [9 9];
```

Calculate the outage probability due to rain attenuation with site diversity.

```
outage = p618SiteDiversityOutage(cfgsd)
```

```
outage = 0.0338
```

References

[1] International Telecommunication Union, ITU-R Recommendation P.618 (12/2017).

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`p618Config`

Functions

`p618PropagationLosses` | `p618SiteDiversityOutage`

Introduced in R2021a

p618Config

Create P.618 configuration object

Description

The `p618Config` object sets the P.618 configuration parameters required for the calculation of the Earth-space propagation losses, cross-polarization discrimination, and sky noise temperature, as defined in the ITU-R P.618 recommendation [1].

Creation

Syntax

```
cfgP618 = p618Config
cfgP618 = p618Config(Name,Value)
```

Description

`cfgP618 = p618Config` creates a P.618 configuration object with default property values.

`cfgP618 = p618Config(Name,Value)` specifies “Properties” on page 3-13 using one or more name-value pair arguments. Enclose each property name in quotes. For example, p618Config(`'GasAnnualExceedance',10,'AntennaEfficiency',0.65`) configures a P.618 configuration object with 10% average annual time percentage of excess for gaseous attenuation and 0.65 antenna efficiency.

Properties

`Frequency` — Signal frequency

`14.25e9` (default) | scalar in the range [1e9, 55e9]

Signal frequency in Hz, specified as a scalar in the range [1e9, 55e9].

Data Types: `double` | `single`

`ElevationAngle` — Elevation angle

`31.0769` (default) | scalar in the range [5, 90]

Elevation angle in degrees, specified as a scalar in the range [5, 90].

Data Types: `double` | `single`

`Latitude` — Earth station latitude

`51.5000` (default) | scalar in the range [-90, 90]

Earth station latitude in degrees, specified as a scalar in the range [-90, 90]. A positive value corresponds to a North latitude, and a negative value corresponds to a South latitude.

Data Types: `double` | `single`

`Longitude` — Earth station longitude
`-0.1400` (default) | scalar in the range [-180, 180]

Earth station longitude in degrees, specified as a scalar in the range [-180, 180]. A positive value corresponds to East longitude, and a negative value corresponds to West longitude.

Data Types: `double` | `single`

`GasAnnualExceedance` — Average annual time percentage of excess for gaseous attenuation
`1` (default) | scalar in the range [0.1, 99]

Average annual time percentage of excess for the gaseous attenuation, specified as a scalar in the range [0.1, 99]. This property calculates the gaseous attenuation, which satisfies the exceedance condition, in terms of the percentage of an average year.

Note The fraction of time during which a preselected threshold is exceeded in an average year is referred to as the annual time percentage of excess.

Data Types: `double` | `single`

`CloudAnnualExceedance` — Average annual time percentage of excess for cloud attenuation
`1` (default) | scalar in the range [0.1, 99]

Average annual time percentage of excess for the cloud attenuation, specified as a scalar in the range [0.1, 99]. This property calculates the cloud attenuation, which satisfies the exceedance condition, in terms of the percentage of an average year.

Data Types: `double` | `single`

`RainAnnualExceedance` — Average annual time percentage of excess for rain attenuation
`1` (default) | scalar in the range [0.001, 5]

Average annual time percentage of excess for the rain attenuation, specified as a scalar in the range [0.001, 5]. This property calculates the rain attenuation, which satisfies the exceedance condition, in terms of the percentage of an average year.

Data Types: `double` | `single`

`ScintillationAnnualExceedance` — Average annual time percentage of excess for tropospheric scintillation
`1` (default) | scalar in the range [0.01, 50]

Average annual time percentage of excess for the tropospheric scintillation, specified as a scalar in the range [0.01, 50]. This property calculates the tropospheric scintillation, which satisfies the exceedance condition, in terms of the percentage of an average year.

Data Types: `double` | `single`

`TotalAnnualExceedance` — Average annual time percentage of excess for total attenuation
`1` (default) | scalar in the range [0.001, 50]

Average annual time percentage of excess for the total attenuation, specified as a scalar in the range [0.001, 50]. This property calculates the total attenuation, which satisfies the exceedance condition, in terms of the percentage of an average year.

Data Types: `double` | `single`

`PolarizationTiltAngle` — Polarization tilt angle
`0` (default) | scalar in the range [-90, 90]

Polarization tilt angle in degrees, specified as a scalar in the range [-90, 90].

Data Types: `double` | `single`

`AntennaDiameter` — Physical diameter of earth station antenna
`1` (default) | positive scalar

Physical diameter of the earth station antenna in meters, specified as a positive scalar.

Data Types: `double` | `single`

`AntennaEfficiency` — Antenna efficiency of earth station antenna
`0.5` (default) | positive scalar

Antenna efficiency of the earth station antenna, specified as a positive scalar.

Data Types: `double` | `single`

Object Functions

Specific to This Object

p618PropagationLosses	Calculate Earth-space propagation losses, cross-polarization discrimination, and sky noise temperature

Examples

Create P.618 Configuration Object

Create a default P.618 configuration object.

```
cfg = p618Config;
```

Specify the signal frequency as 25 GHz, elevation angle as 45 degrees, and antenna efficiency as 0.65. Set the time percentage of excess for the total attenuation per annum as 0.001.

```
cfg.Frequency = 25e9;
cfg.ElevationAngle = 45;
cfg.AntennaEfficiency = 0.65;
cfg.TotalAnnualExceedance = 0.001;
```

Set the earth station direction.

```
cfg.Latitude = 30;   % North direction
cfg.Longitude = 120; % East direction
```

Display the properties of the configuration object.

```
disp(cfg)
```

```
  p618Config with properties:

                      Frequency: 2.5000e+10
                 ElevationAngle: 45
                       Latitude: 30
                      Longitude: 120
            GasAnnualExceedance: 1
          CloudAnnualExceedance: 1
           RainAnnualExceedance: 1
  ScintillationAnnualExceedance: 1
          TotalAnnualExceedance: 1.0000e-03
          PolarizationTiltAngle: 0
                AntennaDiameter: 1
            AntennaEfficiency: 0.6500
```

Calculate Propagation Losses, Cross-Polarization Discrimination, and Sky Noise Temperature

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and unzip the MAT-files.

```
if ~exist('ITURDigitalMaps.tar.gz', 'file')
    url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
    websave('ITURDigitalMaps.tar.gz',url);
    untar('ITURDigitalMaps.tar.gz');
end
```

Create a default P.618 configuration object.

```
cfg = p618Config;
```

Specify the time percentage of excess for the rain attenuation per annum as 0.01 and the time percentage of excess for the total attenuation per annum as 0.001.

```
cfg.RainAnnualExceedance = 0.01;
cfg.TotalAnnualExceedance = 0.001;
```

Calculate the propagation losses, cross-polarization discrimination, and sky noise temperature.

```
[pl,xpd,tsky] = p618PropagationLosses(cfg)
```

```
pl = struct with fields:
    Ag: 0.2269
    Ac: 0.4552
    Ar: 6.7981
    As: 0.2633
    At: 15.6091


xpd = 32.8876

tsky = 267.4689
```

Calculate Propagation Losses in Light Rainfall

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and unzip the MAT-files.

```
if ~exist('ITURDigitalMaps.tar.gz','file')
    url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
    websave('ITURDigitalMaps.tar.gz',url);
    untar('ITURDigitalMaps.tar.gz');
end
```

Create a P.618 configuration object that occupies a signal frequency of 20 GHz.

```
cfg = p618Config('Frequency',20e9);
```

Calculate the propagation losses in a light rainfall of 1 mm/hr with an earth station height of 0.75 km.

```
pl =  p618PropagationLosses(cfg,'RainRate',1,'StationHeight',0.75)
```

```
pl = struct with fields:
    Ag: 0.7996
    Ac: 0.8793
    Ar: 0.0177
    As: 0.3187
    At: 1.7514
```

References

[1] International Telecommunication Union, ITU-R Recommendation P.618 (12/2017).

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`p618SiteDiversityConfig`

Functions

`p618PropagationLosses` | `p618SiteDiversityOutage`

Introduced in R2021a

satelliteScenario

Create satellite scenario object

Description

The `satelliteScenario` object represents a 3-D arena consisting of satellites, ground stations, and the interactions between them. Use this object to model satellite constellations, model ground station networks, perform access analyses between the satellites and the ground stations, and visualize the results.

Creation

Syntax

```
sc = satelliteScenario
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

Description

`sc = satelliteScenario` creates a default satellite scenario object.

`sc = satelliteScenario(startTime,stopTime,sampleTime)` sets the StartTime, StopTime, and SampleTime properties to the values of `startTime`, `stopTime`, and `sampleTime` respectively.

Properties

`StartTime` — Start time of satellite scenario simulation in UTC
current time or earliest epoch defined in TLE data (default) | `datetime` scalar

Start time of the satellite scenario simulation in Universal Time Coordinated (UTC), specified as a `datetime` scalar. If you specify the `StartTime`, `StopTime`, or `SampleTime` properties, the object no longer updates `StartTime` property with further additions of satellites from TLE files.

Example: `datetime(2020,5,11,12,35,38);`

Data Types: `datetime`

`StopTime` — Stop time of satellite scenario simulation in UTC
`StartTime` + longest orbital period among the satellites in the scenario (default) | `datetime` scalar

Stop time of the satellite scenario simulation in UTC, specified as a `datetime` scalar. If you specify the `StartTime`, `StopTime`, or `SampleTime` properties, the object no longer updates `StartTime` property with further additions of satellites from TLE files.

Example: `datetime(2020,5,11,12,35,38);`

Data Types: `datetime`

`SampleTime` — Sample time of satellite scenario simulation
(`StopTime` - `StartTime`)/99 (default) | real-valued scalar

Sample time of the satellite scenario simulation, specified as a real-valued scalar. If you specify the `StartTime`, `StopTime`, or `SampleTime` properties, the object no longer updates, the `SampleTime` property updated with further additions of satellites from TLE files.

Data Types: `double`

`Satellites` — Satellites in the scenario
row vector of `Satellite` objects

This property is read-only.

Satellites in the scenario, returned as a vector of `Satellite` objects. To create a `Satellite` object and add it to the satellite scenario, see the `satellite` object function.

`GroundStations` — Ground stations in scenario
row vector of `GroundStation` objects

This property is read-only.

Ground stations in the scenario, returned as a row vector of `GroundStation` objects. To create a `GroundStation` object and add it to the satellite scenario, see the `groundStation` object function.

`Autoshow` — Graphics shown automatically
`1` or `true` (default) | `0` or `false`

Option to automatically show graphics, specified as a numeric or logical value of `1` (`true`) or `0` (`false`). This property determines if entities added to the scenario are automatically shown in an open `satelliteScenarioViewer`.

Object Functions

groundStation	Add ground station to satellite scenario
satellite	Add satellites to satellite scenario
satelliteScenarioViewer	Create viewer for satellite scenario
play	Play satellite scenario simulation results on viewer

Examples

Create Satellite Scenario with Custom Start and Stop Times

Specify the start time in the current time zone as yesterday. The simulation lasts for half a day.

```
startTime = datetime("yesterday","TimeZone","local");
stopTime = startTime + days(0.5);
```

Specify the sample time as 60 seconds. Create a satellite scenario object, specifying the start time, stop time, and sample time.

```
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc =
  satelliteScenario with properties:

         StartTime: 22-Feb-2021 05:00:00
```

```
          StopTime: 22-Feb-2021 17:00:00
        SampleTime: 60
           Viewers: [0x0 matlabshared.satellitescenario.Viewer]
        Satellites: []
    GroundStations: []
          AutoShow: 1
```

Add Satellites to Scenario Using Keplerian Elements

Create a satellite scenario with a start time of 02-June-2020 8:23:00 AM UTC, and the stop time set to one day later. Set the simulation sample time to 60 seconds.

```
startTime = datetime(2020,6,02,8,23,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add two satellites to the scenario using their Keplerian elements.

```
semiMajorAxis = [10000000; 15000000];
eccentricity = [0.01; 0.02];
inclination = [0; 10];
rightAscensionOfAscendingNode = [0; 15];
argumentOfPeriapsis = [0; 30];
trueAnomaly = [0; 20];

sat = satellite(sc, semiMajorAxis, eccentricity, inclination, ...
    rightAscensionOfAscendingNode, argumentOfPeriapsis, trueAnomaly)
```

```
sat =
  1×2 Satellite array with properties:

    Name
    ID
    ConicalSensors
    Gimbals
    Transmitters
    Receivers
    Accesses
    GroundTrack
    Orbit
    OrbitPropagator
    MarkerColor
    MarkerSize
    ShowLabel
    LabelFontSize
    LabelFontColor
```

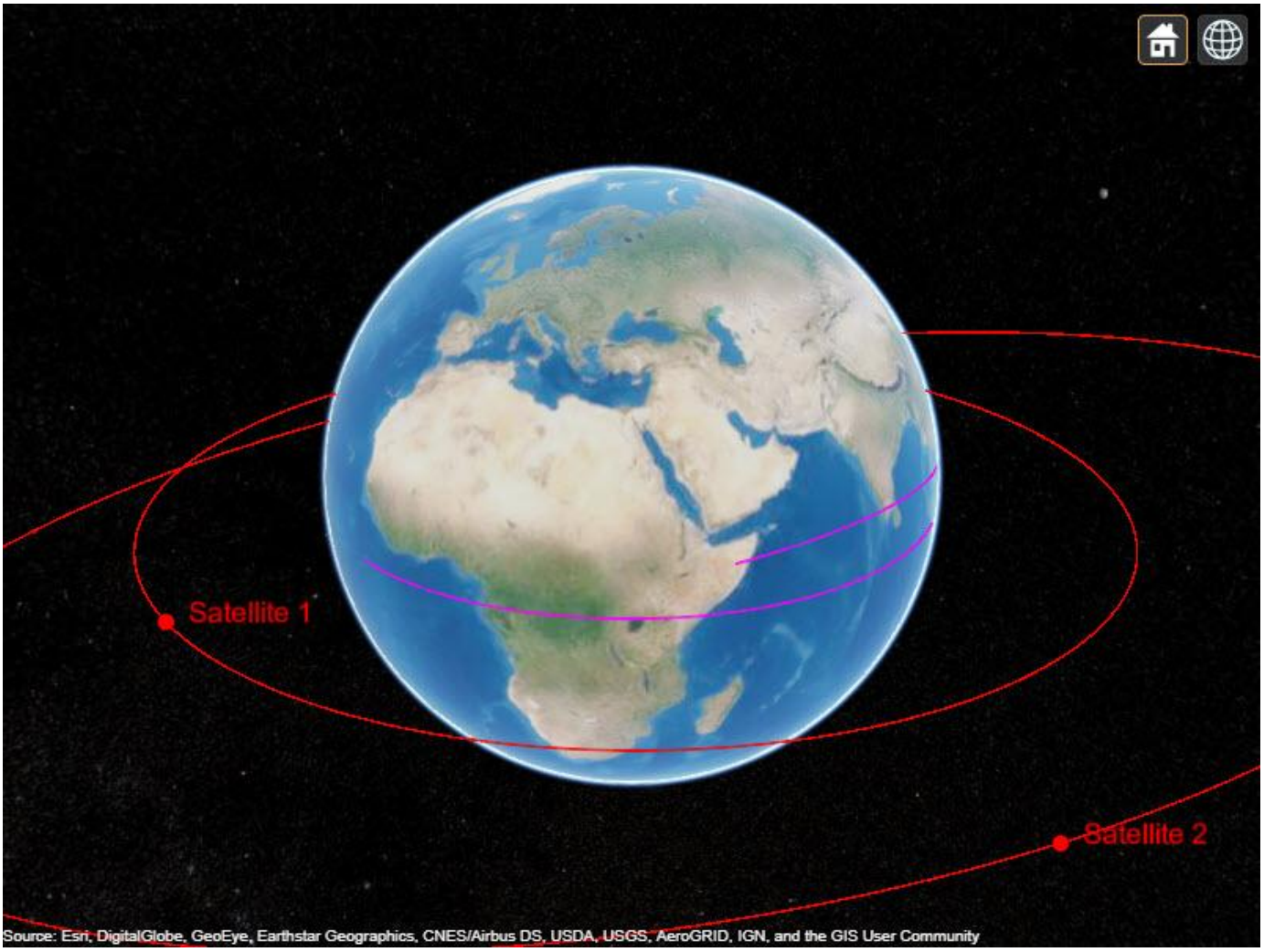
View the satellites in orbit and the ground tracks over one hour.

```
show(sat)
groundTrack(sat,'LeadTime',3600)
```

```
ans=1×2 object
  1×2 GroundTrack array with properties:
```

```
    LeadTime
    TrailTime
    LineWidth
    TrailLineColor
    LeadLineColor
    VisibilityMode

play(sc)
```



Tips

- For correct visualization of all items in the scenario, save either the entire workspace or the scenario containing all the elements.

See Also

Objects
`satellite` | `satelliteScenarioViewer`

Functions
`access` | `groundStation` | `hide` | `play` | `show`

Topics
“Multi-Hop Satellite Communications Link Between Two Ground Stations”
“Satellite Constellation Access to a Ground Station”
“Comparison of Orbit Propagators”
“Modeling Satellite Constellations using Ephemeris Data”
“Estimate GNSS Receiver Position with Simulated Satellite Constellations”
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

skyplot

Plot satellite azimuth and elevation data

Syntax

```
skyplot(azdata,eldata)
skyplot(azdata,eldata,labeldata)
skyplot(status)
skyplot(___,Name,Value)

skyplot(parent,___)
h = skyplot(___)
```

Description

`skyplot(azdata,eldata)` creates a sky plot using the azimuth and elevation data specified as vectors in degrees. Azimuth angles are measured in degrees, clockwise-positive from the North direction. Elevation angles are measured from the horizon line with 90 degrees being directly up. For details about the sky plot figure elements, see “Main Sky Plot Elements” on page 3-29.

`skyplot(azdata,eldata,labeldata)` specifies data labels as a string array with elements corresponding to each data point in the `azdata` and `eldata` inputs.

`skyplot(status)` specifies the azimuth and elevation data in a structure with fields `SatelliteAzimuth` and `SatelliteElevation`.

`skyplot(___,Name,Value)` specifies options using one or more name-value arguments in addition to the input arguments in previous syntaxes. The name-value arguments are properties of the `SkyPlotChart` object. For a list of properties, see SkyPlotChart Properties.

`skyplot(parent,___)` creates the sky plot in the figure, panel, or tab specified by `parent`.

`h = skyplot(___)` returns the sky plot as a `SkyPlotChart` object, `h`. Use `h` to modify the properties of the chart after creating it. For a list of properties, see SkyPlotChart Properties.

Examples

View Satellite Positions from GNSS Sensor

Create a GNSS sensor model as a `gnssSensor` System Object™.

```
gnss = gnssSensor;
```

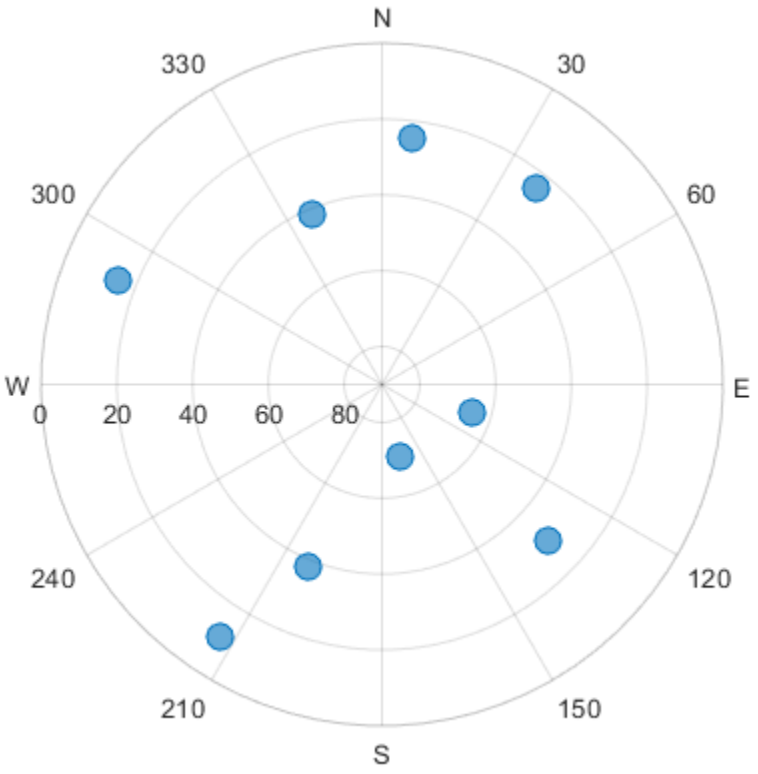
Specify the position and velocity of the sensor. Simulate the sensor readings and get status from visible satellites. Store the azimuth and elevation angles as vectors.

```
pos = [0 0 0];
vel = [0 0 0];
[~, ~, status] = gnss(pos, vel);
```

```
satAz = status.SatelliteAzimuth;
satEl = status.SatelliteElevation;
```

Plot the satellite postions.

```
skyplot(satAz,satEl)
```

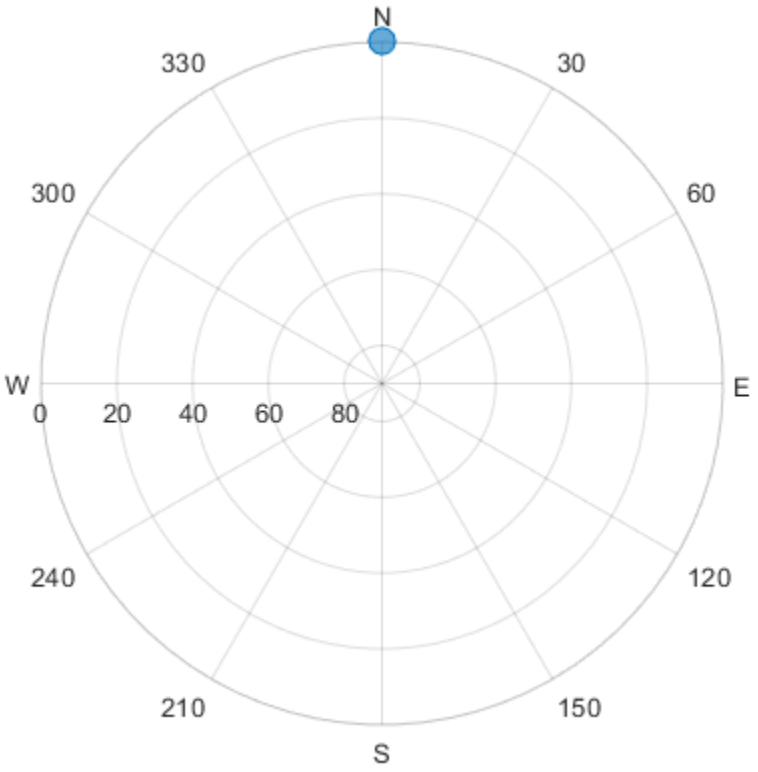


Plot Series of Satellite Positions Over Time

Animate the trajectory of satellite positions over time from a GNSS sensor.

Initialize the sky plot figure. Specify the relevant time-stepping information.

```
skyplotHandle = skyplot(0,0);
```




```
numHours = 12;
dt = 100;
numSeconds = numHours * 60 * 60;
numSimSteps = numSeconds/dt;
```

Create a GNSS sensor model as a `gnssSensor` (Navigation Toolbox) System Object™.

```
gnss = gnssSensor('SampleRate', 1/dt);
```

Iterate through the time steps and do the following:

- Simulate the sensor readings. Specify the zero postion and velocity for the stationary sensor.
- Store the azimuth and elevation angles as vectors.
- Set the `AzimuthData` and `ElevationData` properties of the `SkyPlotChart` handle directly.

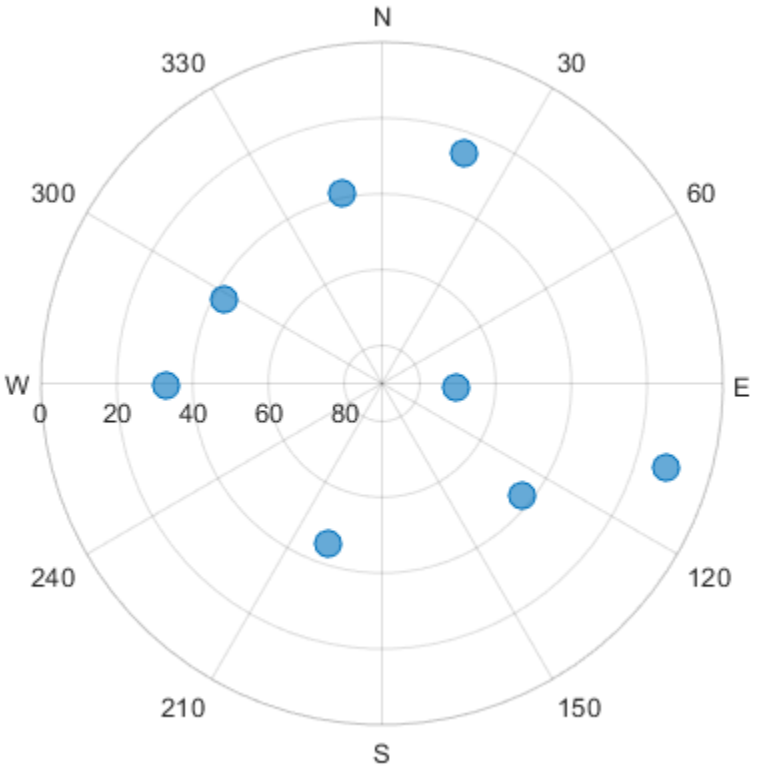
```
for i = 1:numSimSteps

    [~, ~, status] = gnss([0 0 0],[0 0 0]);

    satAz = status.SatelliteAzimuth;
    satEl = status.SatelliteElevation;

    set(skyplotHandle,'AzimuthData',satAz,'ElevationData',satEl);

    drawnow
end
```



View Satellite Positions For Different Groups

Load the azimuth and elevation data from a logfile generated by an Adafruit® GPS satellite sensor. The data provided in this example contains the azimuth and elevation of each satellite and the pseudorandom noise (PRN) codes. Store these values as vectors.

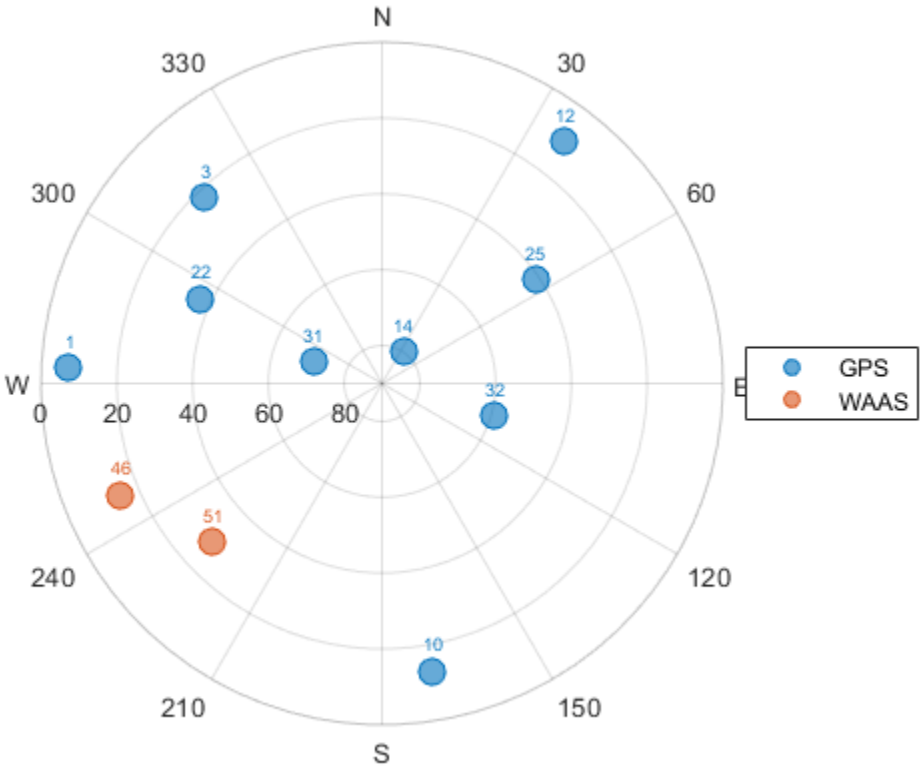
```
load('gpsHWInfo','hwInfo')
satAz = hwInfo.SatelliteAzimuths;
satEl = hwInfo.SatelliteElevations;
prn = hwInfo.SatellitePRNs;
```

Separate the satellites based on the PRN codes. To correlate each position with a group, create a `categorical` array. PRNs greater than 32 are part of the wide area augmentation system (WAAS).

```
isWAAS = (prn > 32);
constellationGroup = categorical(isWAAS,[false,true],{'GPS','WAAS'});
```

Visualize the satellites and specify the categorical groups in the `GroupData` name-value argument. Specify the PRN as the label for each point. Show the legend.

```
skyplot(satAz,satEl,prn,'GroupData',constellationGroup)
legend('GPS','WAAS')
```



Input Arguments

`azdata` — Azimuth angles for visible satellite positions

n-element vector of angles

Azimuth angles for visible satellite positions, specified as an *n*-element vector of angles. *n* is the number of visible satellite positions in the plot. Azimuth angles are measured in degrees, clockwise-positive from the North direction.

Example: `[25 45 182 356]`

Data Types: `double`

`eldata` — Elevation angles for visible satellite positions

n-element vector of angles

Elevation angles for visible satellite positions, specified as an *n*-element vector of angles. *n* is the number of visible satellite positions in the plot. Elevation angles are measured from the horizon line with 90 degrees being directly up.

Example: `[45 90 27 74]`

Data Types: `double`

`labeldata` — Labels for visible satellite positions

n-element string array

Labels for visible satellite positions, specified as an *n*-element string array. *n* is the number of visible satellite positions in the plot.

Example: `["G1" "G11" "G7" "G3"]`

Data Types: `string`

`status` — Satellite status
structure array

Satellite status, specified as a structure array with fields `SatelliteAzimuth` and `SatelliteElevation`. Typically, this status structure comes from a `gnssSensor` object, which simulates satellite positions and velocities.

Example: `gnss = gnssSensor; [~,~,status] = gnss(position,velocity)`

Data Types: `struct`

`parent` — Parent container
`Figure` object | `Panel` object | `Tab` object | `TiledChartLayout` object | `GridLayout` object

Parent container, specified as a `Figure`, `Panel`, `Tab`, `TiledChartLayout`, or `GridLayout` object.

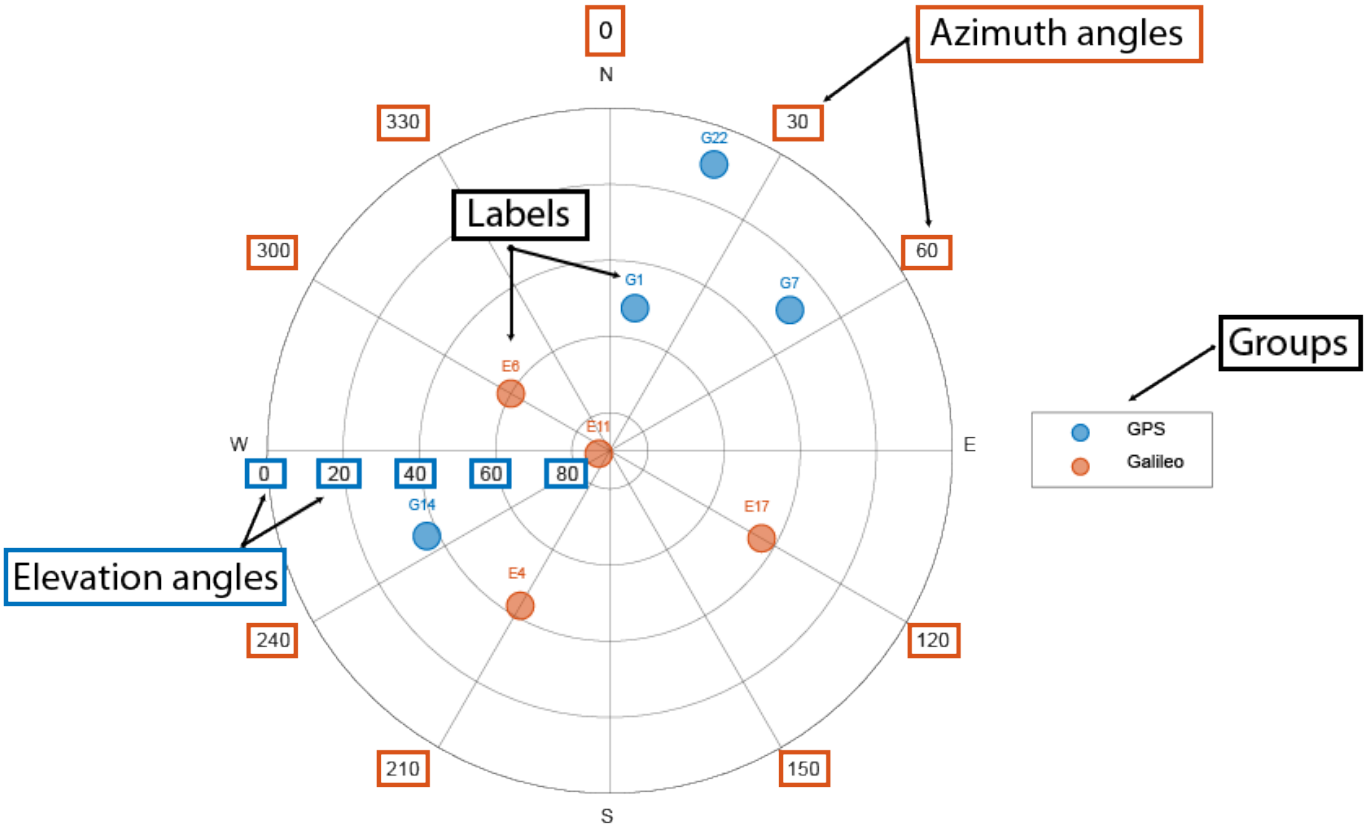
Output Arguments

`h` — Sky plot chart
`SkyplotChart` object

Sky plot chart, returned as a `SkyplotChart` object, which is a standalone visualization on page 3-29. Use `h` to set properties on the sky plot chart. For more information, see SkyPlotChart Properties (Navigation Toolbox).

More About

Main Sky Plot Elements



The main elements of the figure are:

- Azimuth axes — Specified by the `azdata` input argument, azimuth angle positions are measured clockwise-positive from the North direction.
- Elevation axes —Specified by the `eldata` input argument, elevation angle positions are measured from the horizon line with 90 degrees being directly up.
- Labels — Specified by the `labeldata` input argument as a string array with an element for each point in the `azdata` and `eldata` vectors.
- Groups — Specified by the GroupData property, a `categorical` array defines the group for each satellite position.

Standalone Visualization

A standalone visualization is a chart designed for a special purpose that works independently from other charts. Unlike other charts such as `plot` and `surf`, a standalone visualization has a preconfigured axes object built into it, and some customizations are not available. A standalone visualization also has these characteristics:

- It cannot be combined with other graphics elements, such as lines, patches, or surfaces. Thus, the `hold` command is not supported.

- The `gca` function can return the chart object as the current axes.
- You can pass the chart object to many MATLAB functions that accept an axes object as an input argument. For example, you can pass the chart object to the `title` function.

See Also

Functions
`polarscatter`

Properties
SkyPlotChart Properties (Navigation Toolbox)

Objects
`gnssSensor` | `nmeaParser`

Introduced in R2021a

SkyPlotChart Properties

Sky plot chart appearance and behavior

Description

The `SkyPlotChart` properties control the appearance of a sky plot chart generated using the `skyplot` function. To modify the chart appearance, use dot notation on the `SkyPlotChart` object:

```
h = skyplot;
h.AzimuthData = [45 120 295];
h.ElevationData = [10 45 60];
h.Labels = ["G1" "G4" "G11"];
```

Properties

Sky Plot Properties

`AzimuthData` — Azimuth angles for visible satellite positions

n-element vector of angles

Azimuth angles for visible satellite positions, specified as an *n*-element vector of angles. *n* is the number of visible satellite positions in the plot. Angles are measured in degrees, clockwise-positive from the North direction.

Example: `[25 45 182 356]`

Data Types: `double`

`ElevationData` — Elevation angles for visible satellite positions

n-element vector of angles

Elevation angles for visible satellite positions, specified as an *n*-element vector of angles. *n* is the number of visible satellite positions in the plot. Angles are measured from the horizon line with 90 degrees being directly up.

Example: `[45 90 27 74]`

Data Types: `double`

`LabelData` — Labels for visible satellite positions

n-element string array

Labels for visible satellite positions, specified as an *n*-element string array. *n* is the number of visible satellite positions in the plot.

Example: `["G1" "G11" "G7" "G3"]`

Data Types: `string`

`GroupData` — Group for each satellite position

`categorical` array

Group for each satellite position, specified as a `categorical` array. Each group has a different color label defined by the ColorOrder property.

Example: `[GPS GPS Galileo Galileo]`

Data Types: `double`

`ColorOrder` — Color order

seven predefined colors (default) | three-column matrix of RGB triplets

Color order, specified as a three-column matrix of RGB triplets. This property defines the palette of colors MATLAB uses to create plot objects such as `Line`, `Scatter`, and `Bar` objects. Each row of the array is an RGB triplet. An RGB triplet is a three-element vector whose elements specify the intensities of the red, green, and blue components of a color. The intensities must be in the range [0, 1]. This table lists the default colors.

Colors	`ColorOrder` Matrix

	`[ 0 0.4470 0.7410`
	`0.8500 0.3250 0.0980`
	`0.9290 0.6940 0.1250`
	`0.4940 0.1840 0.5560`
	`0.4660 0.6740 0.1880`
	`0.3010 0.7450 0.9330`
	`0.6350 0.0780 0.1840]`

MATLAB assigns colors to objects according to their order of creation. For example, when plotting lines, the first line uses the first color, the second line uses the second color, and so on. If there are more lines than colors, then the cycle repeats.

You can also set the color order using the `colororder` function.

Label Properties

`LabelFontSize` — Font size of labels

scalar numeric value

Font size of labels, specified as a scalar numeric value. The default font depends on the specific operating system and locale.

Example: `h = skyplot(__,'LabelFontSize',12)`

Example: `h.LabelFontSize = 12`

`LabelFontSizeMode` — Selection mode for font size of labels

`'auto'` (default) | `'manual'`

Selection mode for the font size of labels, specified as one of these values:

- `'auto'` — Font size specified by MATLAB. If you resize the axes to be smaller than the default size, the font size can scale down to improve readability and layout.
- `'manual'` — Font size specified manually. MATLAB does not scale the font size as the axes size changes. To specify the font size, set the `LabelFontSize` property.

Chart Properties

`HandleVisibility` — Visibility of object handle

`'on'` (default) | `'off'` | `'callback'`

Visibility of the `SkyPlotChart` object handle in the `Children` property of the parent, specified as one of these values:

- `'on'` — Object handle is always visible.
- `'off'` — Object handle is invisible at all times. This option is useful for preventing unintended changes to the UI by another function. To temporarily hide the handle during the execution of that function, set the `HandleVisibility` to `'off'`.
- `'callback'` — Object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line. This option blocks access to the object at the command line, but allows callback functions to access it.

If the object is not listed in the `Children` property of the parent, then functions that obtain object handles by searching the object hierarchy or querying handle properties cannot return it. This includes `get`, `findobj`, `gca`, `gcf`, `gco`, `newplot`, `cla`, `clf`, and `close`.

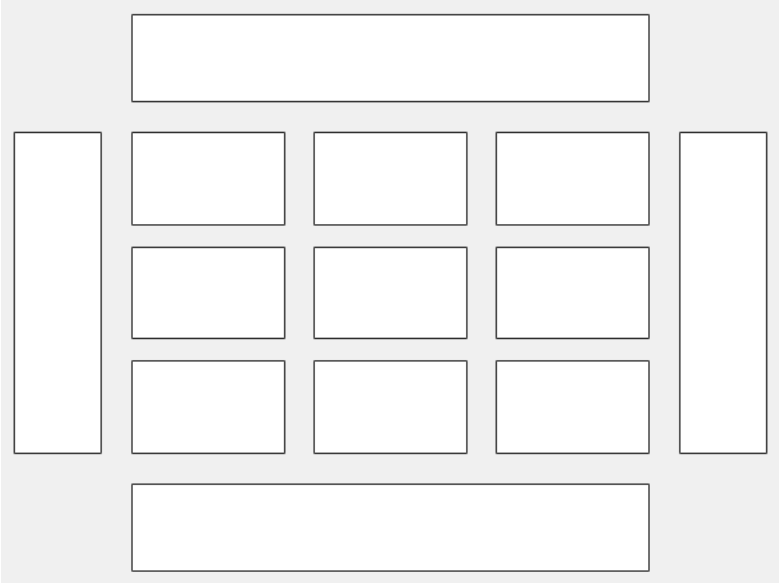
Hidden object handles are still valid. Set the root `ShowHiddenHandles` property to `'on'` to list all object handles, regardless of their `HandleVisibility` property setting.

`Layout` — Layout options

empty `LayoutOptions` array (default) | `TiledChartLayoutOptions` object | `GridLayoutOptions` object

Layout options, specified as a `TiledChartLayoutOptions` or `GridLayoutOptions` object. This property is useful when the chart is either in a tiled chart layout or a grid layout.

To position the chart within the grid of a tiled chart layout, set the `Tile` and `TileSpan` properties on the `TiledChartLayoutOptions` object. For example, consider a 3-by-3 tiled chart layout. The layout has a grid of tiles in the center, and four tiles along the outer edges. In practice, the grid is invisible and the outer tiles do not take up space until you populate them with axes or charts.



This code places the chart `c` in the third tile of the grid..

```
c.Layout.Tile = 3;
```

To make the chart span multiple tiles, specify the `TileSpan` property as a two-element vector. For example, this chart spans 2 rows and 3 columns of tiles.

```
c.Layout.TileSpan = [2 3];
```

To place the chart in one of the surrounding tiles, specify the `Tile` property as `'north'`, `'south'`, `'east'`, or `'west'`. For example, setting the value to `'east'` places the chart in the tile to the right of the grid.

```
c.Layout.Tile = 'east';
```

To place the chart into a layout within an app, specify this property as a `GridLayoutOptions` object. For more information about working with grid layouts in apps, see `uigridlayout`.

If the chart is not a child of either a tiled chart layout or a grid layout (for example, if it is a child of a figure or panel) then this property is empty and has no effect.

`Parent` — Parent container

`Figure` object | `Panel` object | `Tab` object | `TiledChartLayout` object | `GridLayout` object

Parent container, specified as a `Figure`, `Panel`, `Tab`, `TiledChartLayout`, or `GridLayout` object.

Marker Properties

`MarkerEdgeAlpha` — Marker edge transparency

1 (default) | scalar in range `[0,1]` | `'flat'`

Marker edge transparency, specified as a scalar in the range `[0,1]` or `'flat'`. A value of 1 is opaque and 0 is completely transparent. Values between 0 and 1 are semitransparent.

To set the edge transparency to a different value for each point in the plot, set the `AlphaData` property to a vector the same size as the `XData` property, and set the `MarkerEdgeAlpha` property to `'flat'`.

`MarkerEdgeColor` — Marker outline color

`'flat'` (default) | `'auto'` | RGB triplet | hexadecimal color code | `'r'` | `'g'` | `'b'` | ...

Marker outline color, specified as `'auto'`, an RGB triplet, a hexadecimal color code, a color name, or a short name. The value of `'auto'` uses the same color as the `Color` property.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`. For example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and the hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'red'	'r'	[1 0 0]	'#FF0000'	
'green'	'g'	[0 1 0]	'#00FF00'	
'blue'	'b'	[0 0 1]	'#0000FF'	
'cyan'	'c'	[0 1 1]	'#00FFFF'	
'magenta'	'm'	[1 0 1]	'#FF00FF'	
'yellow'	'y'	[1 1 0]	'#FFFF00'	
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	
'none'	Not applicable	Not applicable	Not applicable	No color

This table shows the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

MarkerFaceAlpha — Marker face transparency

0.6 (default) | scalar in range [0,1] | 'flat'

Marker face transparency, specified as a scalar in the range [0,1] or 'flat'. A value of 1 is opaque and 0 is completely transparent. Values between 0 and 1 are partially transparent.

To set the marker face transparency to a different value for each point, set the AlphaData property to a vector the same size as the XData property, and set the MarkerFaceAlpha property to 'flat'.

MarkerFaceColor — Marker fill color

'flat' (default) | 'auto' | 'none' | RGB triplet | hexadecimal color code | 'r' | 'g' | 'b' | ...

Marker fill color, specified as 'flat', 'auto', an RGB triplet, a hexadecimal color code, a color name, or a short name. The 'flat' option uses the CData values. The 'auto' option uses the same color as the Color property for the axes.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].

- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	
`[0.8500 0.3250 0.0980]`	`'#D95319'`	
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	

Example: `[0.3 0.2 0.1]`

Example: `'green'`

Example: `'#D2F9A7'`

`MarkerSizeData` — Marker size

`100` (default) | positive scalar | vector of positive values

Marker size, specified as a positive scalar or vector of positive values in points, where one point = 1/72 of an inch. If specified as a vector, the vector must be of the same length as `AzimuthData`.

Position

`PositionConstraint` — Position to hold constant

`'outerposition'` | `'innerposition'`

Position property to hold constant when adding, removing, or changing decorations, specified as one of the following values:

- `'outerposition'` — The `OuterPosition` property remains constant when you add, remove, or change decorations such as a title or an axis label. If any positional adjustments are needed, MATLAB adjusts the `InnerPosition` property.
- `'innerposition'` — The `InnerPosition` property remains constant when you add, remove, or change decorations such as a title or an axis label. If any positional adjustments are needed, MATLAB adjusts the `OuterPosition` property.

Note Setting this property has no effect when the parent container is a `TiledChartLayout`.

`OuterPosition` — Outer size and location

`[0 0 1 1]` (default) | four-element vector

Outer size and location of the skyplot within the parent container (typically a figure, panel, or tab), specified as a four-element vector of the form `[left bottom width height]`. The outer position includes the colorbar, title, and axis labels.

- The `left` and `bottom` elements define the distance from the lower-left corner of the container to the lower-left corner of the skyplot.
- The `width` and `height` elements are the skyplot dimensions, which include the skyplot cells, plus a margin for the surrounding text and colorbar.

The default value of `[0 0 1 1]` covers the whole interior of the container. The units are normalized relative to the size of the container. To change the units, set the `Units` property.

Note Setting this property has no effect when the parent container is a `TiledChartLayout`.

`InnerPosition` — Inner size and location

`[0.1300 0.1100 0.7750 0.8114]` (default) | four-element vector

Inner size and location of the skyplot within the parent container (typically a figure, panel, or tab), specified as a four-element vector of the form `[left bottom width height]`. The inner position does not include the colorbar, title, or axis labels.

- The `left` and `bottom` elements define the distance from the lower-left corner of the container to the lower-left corner of the skyplot.
- The `width` and `height` elements are the skyplot dimensions, which include only the skyplot cells.

Note Setting this property has no effect when the parent container is a `TiledChartLayout`.

`Position` — Inner size and location

four-element vector

Inner size and location of the skyplot within the parent container (typically a figure, panel, or tab), specified as a four-element vector of the form `[left bottom width height]`. This property is equivalent to the `InnerPosition` property.

Note Setting this property has no effect when the parent container is a `TiledChartLayout`.

`Units` — Position units

`'normalized'` (default) | `'inches'` | `'centimeters'` | `'points'` | `'pixels'` | `'characters'`

Position units, specified as one of these values.

`Units`	Description
`'normalized'` (default)	Normalized with respect to the container, which is typically the figure or a panel. The lower left corner of the container maps to `(0,0)`, and the upper right corner maps to `(1,1)`.
`'inches'`	Inches.
`'centimeters'`	Centimeters.
`'characters'`	Based on the default `uicontrol` font of the graphics root object: • Character width = width of letter `x`. • Character height = distance between the baselines of two lines of text.
`'points'`	Typography points. One point equals 1/72 inch.
`'pixels'`	Pixels. Starting in R2015b, distances in pixels are independent of your system resolution on Windows® and Macintosh systems: • On Windows systems, a pixel is 1/96th of an inch. • On Macintosh systems, a pixel is 1/72nd of an inch. On Linux® systems, the size of a pixel is determined by your system resolution.

When specifying the units as a name-value argument during object creation, you must set the `Units` property before specifying the properties that you want to use these units, such as `OuterPosition`.

`Visible` — State of visibility

`'on'` (default) | on/off logical value

State of visibility, specified as `'on'` or `'off'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

- `'on'` — Display the skyplot.
- `'off'` — Hide the skyplot without deleting it. You can still access the properties of an invisible `SkyPlotChart` object.

See Also

Functions
`polarscatter` | `skyplot`

Objects
`gnssSensor` | `nmeaParser`

Introduced in R2021a

Satellite

Satellite object belonging to satellite scenario

Description

Satellite defines a satellite object belonging to a satellite scenario.

Creation

You can create Satellite objects using the `satellite` method of `satelliteScenario`.

Properties

`Orbit` — Orbit graphic
`Orbit` object

Orbit object parameters for a satellite, specified as an `orbit` object. Only these object properties are relevant for this function.

`LineColor` — Color of orbit
`[1,0,0]` (default) | RGB triplet | hexadecimal color code | `'r'` | `'g'` | `'b'`

Color of the orbit, specified as an RGB triplet, hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	
'none'	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

Example: 'blue'

Example: [0 0 1]

Example: '#0000FF'

LineWidth — Visual width of orbit

1 (default) | scalar in the range (0, 10)

Visual width of orbit in pixels, specified as a scalar in the range (0, 10).

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

VisibilityMode — Visibility mode of orbit graphic

'inherit' (default) | 'manual'

Visibility mode of orbit graphic, specified as one of these values:

- 'inherit' — Visibility of the graphic matches that of the parent
- 'manual' — Visibility of the graphic is not inherited and is independent of that of the parent

Data Types: char | string

Accesses — Access analysis objects

row vector of Access objects

You can set this property only when calling Satellite. After you call Satellite, this property is read-only.

Access analysis objects, specified as a row vector of Access objects.

`MarkerColor` — Color of marker

`[1 0 0]` (default) | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the marker, specified as a comma-separated pair consisting of `'MarkerColor'` and either an RGB triplet or a string or character vector of a color name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	
`[0.8500 0.3250 0.0980]`	`'#D95319'`	
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	

`MarkerSize` — Size of marker

`10` (default) | positive scalar less than 30

Size of the marker, specified as a comma-separated pair consisting of `'MarkerSize'` and a real positive scalar less than 30. The unit is in pixels.

`ShowLabel` — State of Satellite label visibility
`true` or `1` (default) | `false` or `0`

State of Satellite label visibility, specified as a comma-separated pair consisting of `'ShowLabel'` and numerical or logical value of `1` (`true`) or `0` (`false`).

Data Types: `logical`

`LabelFontSize` — Font size of Satellite label
15 (default) | positive scalar less than 30

Font size of the Satellite label, specified as a comma-separated pair consisting of `'LabelFontSize'` and a positive scalar less than `30`.

`LabelFontColor` — Font color of Satellite label
`[1,0,0]` (default) | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Font color of the Satellitelabel, specified as a comma-separated pair consisting of `'LabelFontColor'` and either an RGB triplet or a string or character vector of a color name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

`Name` — Satellite name
`"Satellite idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling Satellite. After you call Satellite, this property is read-only.

Satellite name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one Satellite is added, specify `Name` as a string scalar or a character vector.
- If multiple Satellites are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the Satellite added by the `Satellite` object function. If another Satellite of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`ID` — Satellite ID assigned by simulator
real positive scalar

This property is set internally by the simulator and is read-only.

Satellite ID assigned by the simulator, specified as a positive scalar.

`ConicalSensors` — Conical sensors
row vector of conical sensors

You can set this property only when calling `conicalSensor`. After you call `conicalSensor`, this property is read-only.

Conical sensors attached to the Satellite, specified as a row vector of conical sensors.

`Gimbals` — Gimbals
row vector of `Gimbal` objects

You can set this property only when calling `gimbal`. After you call `gimbal`, this property is read-only.

Gimbals attached to the Satellite, specified as the comma-separated pair consisting of `'Gimbals'` and a row vector of `Gimbal` objects.

`OrbitPropagator` — Name of orbit propagator

`"sgp4"` (default) | `"two-body-keplerian"` | `"sdp4"` | `"ephemeris"`

You can set this property when calling `satellite` only. After you call `satellite`, this property is read-only.

Name of the orbit propagator used for propagating satellite position and velocity, specified as the comma-separated pair consisting of `'OrbitPropagator'` and either `"two-body-keplerian"`, `"sgp4"`, `"sdp4"`, or `"ephemeris"`.

Dependencies

`OrbitPropagator` is not available for ephemeris data inputs (`timetable` or `timeseries`). In these cases, `satellite` ignores this name-value pair.

Data Types: `string` | `char`

`Receivers` — Receivers attached to Satellite

row vector of `Receiver` objects

You can set this property only when calling `receiver`. After you call `receiver`, this property is read-only.

Receivers attached to the Satellite, specified as a row vector of `Receiver` objects.

`Transmitters` — Transmitters attached to Satellite

row vector of `Transmitter` objects

You can set this property only when calling `transmitter`. After you call `transmitter`, this property is read-only.

Transmitters attached to the Satellite, specified as a row vector of `Transmitter` objects.

`GroundTrack` — Ground track of the Satellite

row vector of `GroundTrack` objects

You can set this property only when calling `groundTrack`. After you call `groundTrack`, this property is read-only.

Ground track of the Satellite, specified as a row vector of `GroundTrack` objects.

Object Functions

access	Add access analysis objects to satellite scenario
states	Position and velocity of satellite
conicalSensor	Add conical sensor to satellite scenario
pointAt	Target at which entity must be pointed
transmitter	Add transmitter to satellite scenario
gimbal	Add gimbal to satellite or ground station
receiver	Add receiver to satellite scenario
show	Show object in satellite scenario viewer
aer	Calculate azimuth angle, elevation angle, and range in NED frame from another satellite or ground station
hide	Hides satellite scenario entity from viewer
groundTrack	Add ground track object to satellite in scenario

orbitalElements Orbital elements of satellites in scenario

See Also

Objects
`access` | `groundStation` | `satelliteScenario` | `satelliteScenarioViewer`

Functions
`hide` | `play` | `show`

Topics
“Multi-Hop Satellite Communications Link Between Two Ground Stations”
“Satellite Constellation Access to a Ground Station”
“Comparison of Orbit Propagators”
“Modeling Satellite Constellations using Ephemeris Data”
“Estimate GNSS Receiver Position with Simulated Satellite Constellations”
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

GroundStation

Ground station object belonging to satellite scenario

Description

The `GroundStation` object defines a ground station object belonging to a satellite scenario.

Creation

You can create `GroundStation` object using the `groundStation` object function of the `satelliteScenario` object.

Properties

`Name` — GroundStation name
`"GroundStation idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling GroundStation. After you call GroundStation, this property is read-only.

GroundStation name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one GroundStation is added, specify `Name` as a string scalar or a character vector.
- If multiple GroundStations are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the GroundStation added by the `GroundStation` object function. If another GroundStation of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`ID` — GroundStation ID assigned by simulator
real positive scalar

This property is set internally by the simulator and is read-only.

GroundStation ID assigned by the simulator, specified as a positive scalar.

`Latitude` — Geodetic latitude of ground stations
`42.3001` (default) | scalar | row vector

You can set this property only when calling GroundStation. After you call GroundStation, this property is read-only.

Geodetic latitude of ground stations, specified as a scalar. Values must be in the range [-90, 90].

- If you add only one ground station, specify Latitude as a scalar double.
- If you add multiple ground stations, specify Latitude as a vector double whose length is equal to the number of ground stations being added.

When latitude and longitude are specified as `lat, lon` inputs to GroundStation, Latitude specified as a name-value argument takes precedence.

Data Types: `double`

`Longitude` — Geodetic longitude of ground stations
`-71.3504` (default) | scalar | row vector

You can set this property only when calling GroundStation. After you call GroundStation, this property is read-only.

Geodetic longitude of ground stations, specified as a scalar or a vector. Values must be in the range [-180, 180].

- If you add only one ground station, specify longitude as a scalar.
- If you add multiple ground stations, specify longitude as a vector whose length is equal to the number of ground stations being added.

When longitude and longitude are specified as `lat, lon` inputs to GroundStation, longitude specified as a name-value argument takes precedence.

Data Types: `double`

`Altitude` — Altitude of ground station
`0` m (default) | scalar | vector

You can set this property only when calling GroundStation. After you call GroundStation, this property is read-only.

Altitude of ground stations, specified as a scalar or a vector.

- If you specify `Altitude` as a scalar, the value is assigned to each ground station in the GroundStation.
- If you specify `Altitude` as a vector, the vector length must be equal to the number of ground stations in the GroundStation.

When latitude and longitude are specified as `lat, lon` inputs to GroundStation, Latitude specified as a name-value argument takes precedence.

Data Types: `double`

`MinElevationAngle` — Minimum elevation angle
`0` (default) | scalar | vector

Minimum elevation angle of a satellite for the satellite to be visible from the ground station, specified as a scalar or row vector. Values must be in the range [–90, 90]. For access and link closure to be possible, the elevation angle must be at least equal to the value specified in `MinElevationAngle`.

- If you specify `MinElevationAngle` as a scalar, the value is assigned to each ground station in the GroundStation.

- If you specify `MinElevationAngle` as a vector, the vector length must be equal to the number of ground stations in the GroundStation.

Data Types: `double`

`Accesses` — Access analysis objects
row vector of `Access` objects

You can set this property only when calling GroundStation. After you call GroundStation, this property is read-only.

Access analysis objects, specified as a row vector of `Access` objects.

`ConicalSensors` — Conical sensors
row vector of conical sensors

You can set this property only when calling `conicalSensor`. After you call `conicalSensor`, this property is read-only.

Conical sensors attached to the GroundStation, specified as a row vector of conical sensors.

`Gimbals` — Gimbals
row vector of `Gimbal` objects

You can set this property only when calling `gimbal`. After you call `gimbal`, this property is read-only.

Gimbals attached to the GroundStation, specified as the comma-separated pair consisting of `'Gimbals'` and a row vector of `Gimbal` objects.

`Transmitters` — Transmitters attached to GroundStation
row vector of `Transmitter` objects

You can set this property only when calling `transmitter`. After you call `transmitter`, this property is read-only.

Transmitters attached to the GroundStation, specified as a row vector of `Transmitter` objects.

`Receivers` — Receivers attached to GroundStation
row vector of `Receiver` objects

You can set this property only when calling `receiver`. After you call `receiver`, this property is read-only.

Receivers attached to the GroundStation, specified as a row vector of `Receiver` objects.

`MarkerColor` — Color of marker
`[1 0 0]` (default) | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the marker, specified as a comma-separated pair consisting of `'MarkerColor'` and either an RGB triplet or a string or character vector of a color name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	
`[0.8500 0.3250 0.0980]`	`'#D95319'`	
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	

`MarkerSize` — Size of marker
`10` (default) | positive scalar less than 30

Size of the marker, specified as a comma-separated pair consisting of `'MarkerSize'` and a real positive scalar less than 30. The unit is in pixels.

`ShowLabel` — State of GroundStation label visibility
`true` or `1` (default) | `false` or `0`

State of GroundStation label visibility, specified as a comma-separated pair consisting of `'ShowLabel'` and numerical or logical value of `1` (`true`) or `0` (`false`).

Data Types: `logical`

`LabelFontSize` — Font size of GroundStation label
15 (default) | positive scalar less than 30

Font size of the GroundStation label, specified as a comma-separated pair consisting of `'LabelFontSize'` and a positive scalar less than `30`.

`LabelFontColor` — Font color of GroundStation label
`[1,0,0]` (default) | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Font color of the GroundStationlabel, specified as a comma-separated pair consisting of `'LabelFontColor'` and either an RGB triplet or a string or character vector of a color name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	
`[0.8500 0.3250 0.0980]`	`'#D95319'`	
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	

RGB Triplet	Hexadecimal Color Code	Appearance
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

Object Functions

access	Add access analysis objects to satellite scenario
conicalSensor	Add conical sensor to satellite scenario
transmitter	Add transmitter to satellite scenario
receiver	Add receiver to satellite scenario
gimbal	Add gimbal to satellite or ground station
show	Show object in satellite scenario viewer
aer	Calculate azimuth angle, elevation angle, and range in NED frame from another satellite or ground station
hide	Hides satellite scenario entity from viewer

Examples

Add Ground stations to Scenario and Visualize Access Intervals

Create satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020, 5, 1, 11, 36, 0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime, stopTime, sampleTime);
lat = [10];
lon = [-30];
gs = groundStation(sc, lat, lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
trueAnomaly = 0;
sat = satellite(sc, semiMajorAxis, eccentricity, inclination, ...
        rightAscensionOfAscendingNode, argumentOfPeriapsis, trueAnomaly);
```

Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

```
ac = access(sat, gs);
intvls = accessIntervals(ac)
```

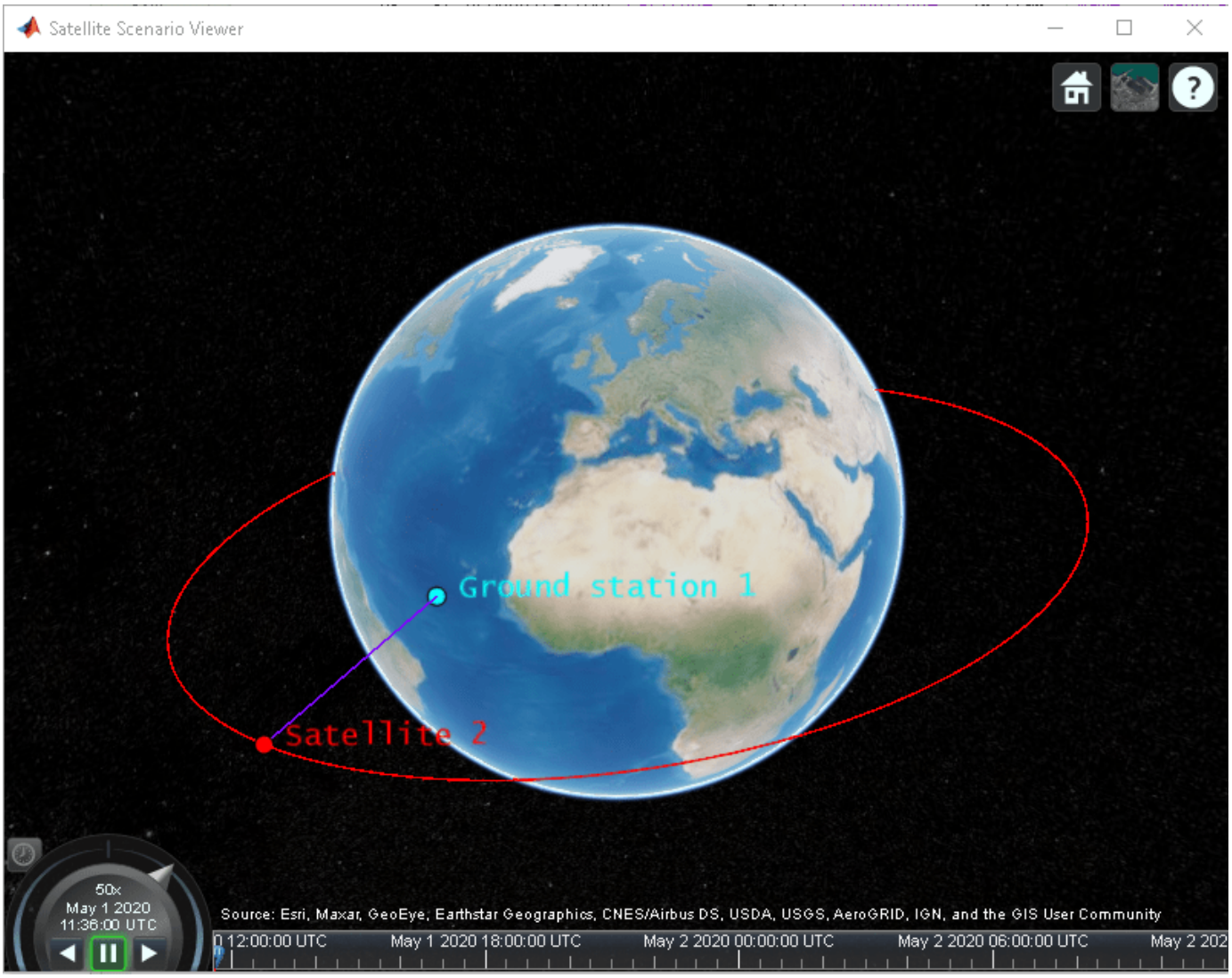
```
intvls=8×8 table
       Source              Target          IntervalNumber         StartTime                EndTi
    _____________    ___________________    _______________    ____________________    ___________

    "Satellite 2"    "Ground station 1"          1           01-May-2020 11:36:00    01-May-2020
```

```
    "Satellite 2"    "Ground station 1"          2          01-May-2020 14:20:00    01-May-2020
    "Satellite 2"    "Ground station 1"          3          01-May-2020 17:27:00    01-May-2020
    "Satellite 2"    "Ground station 1"          4          01-May-2020 20:34:00    01-May-2020
    "Satellite 2"    "Ground station 1"          5          01-May-2020 23:41:00    02-May-2020
    "Satellite 2"    "Ground station 1"          6          02-May-2020 02:50:00    02-May-2020
    "Satellite 2"    "Ground station 1"          7          02-May-2020 05:59:00    02-May-2020
    "Satellite 2"    "Ground station 1"          8          02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `satellite` | `show` | `transmitter`

Topics
“Multi-Hop Satellite Communications Link Between Two Ground Stations”
“Satellite Constellation Access to a Ground Station”
“Comparison of Orbit Propagators”
“Modeling Satellite Constellations using Ephemeris Data”
“Estimate GNSS Receiver Position with Simulated Satellite Constellations”
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

Access

Access analysis object belonging to scenario

Description

The Access object defines an access analysis object belonging to a `Satellite`, `GroundStation` or `ConicalSensor`.

Creation

You can create an `Access` object using the `access` object function of `GroundStation` or `Satellite`.

Properties

`Sequence` — Satellite, ground station, or conical sensor ID

row vector of positive real numbers

You can set this property only when calling `access`. After you call `access`, this property is read-only.

Satellite, ground station, or conical sensor ID defining the nodes of access analysis.

`LineWidth` — Visual width of access analysis object

`1` (default) | scalar

Visual width of access analysis object in pixels, specified as a scalar in the range (0, 10).

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LineColor` — Color of analysis line

`[0.5 0 1]` (default) | RGB triplet | hexadecimal color code | color name | short name

Color of access analysis line, specified as an RGB triplet, hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'red'	'r'	[1 0 0]	'#FF0000'	
'green'	'g'	[0 1 0]	'#00FF00'	
'blue'	'b'	[0 0 1]	'#0000FF'	
'cyan'	'c'	[0 1 1]	'#00FFFF'	
'magenta'	'm'	[1 0 1]	'#FF00FF'	
'yellow'	'y'	[1 1 0]	'#FFFF00'	
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	
'none'	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

Object Functions

show	Show object in satellite scenario viewer
accessStatus	Status of access between first and last node defining access analysis
accessIntervals	Intervals during which access status is true
accessPercentage	Percentage of time when access exists between first and last node defining access analysis
hide	Hides satellite scenario entity from viewer

Examples

Add Ground stations to Scenario and Visualize Access Intervals

Create satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020, 5, 1, 11, 36, 0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime, stopTime, sampleTime);
lat = [10];
lon = [-30];
gs = groundStation(sc, lat, lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
trueAnomaly = 0;
sat = satellite(sc, semiMajorAxis, eccentricity, inclination, ...
        rightAscensionOfAscendingNode, argumentOfPeriapsis, trueAnomaly);
```

Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

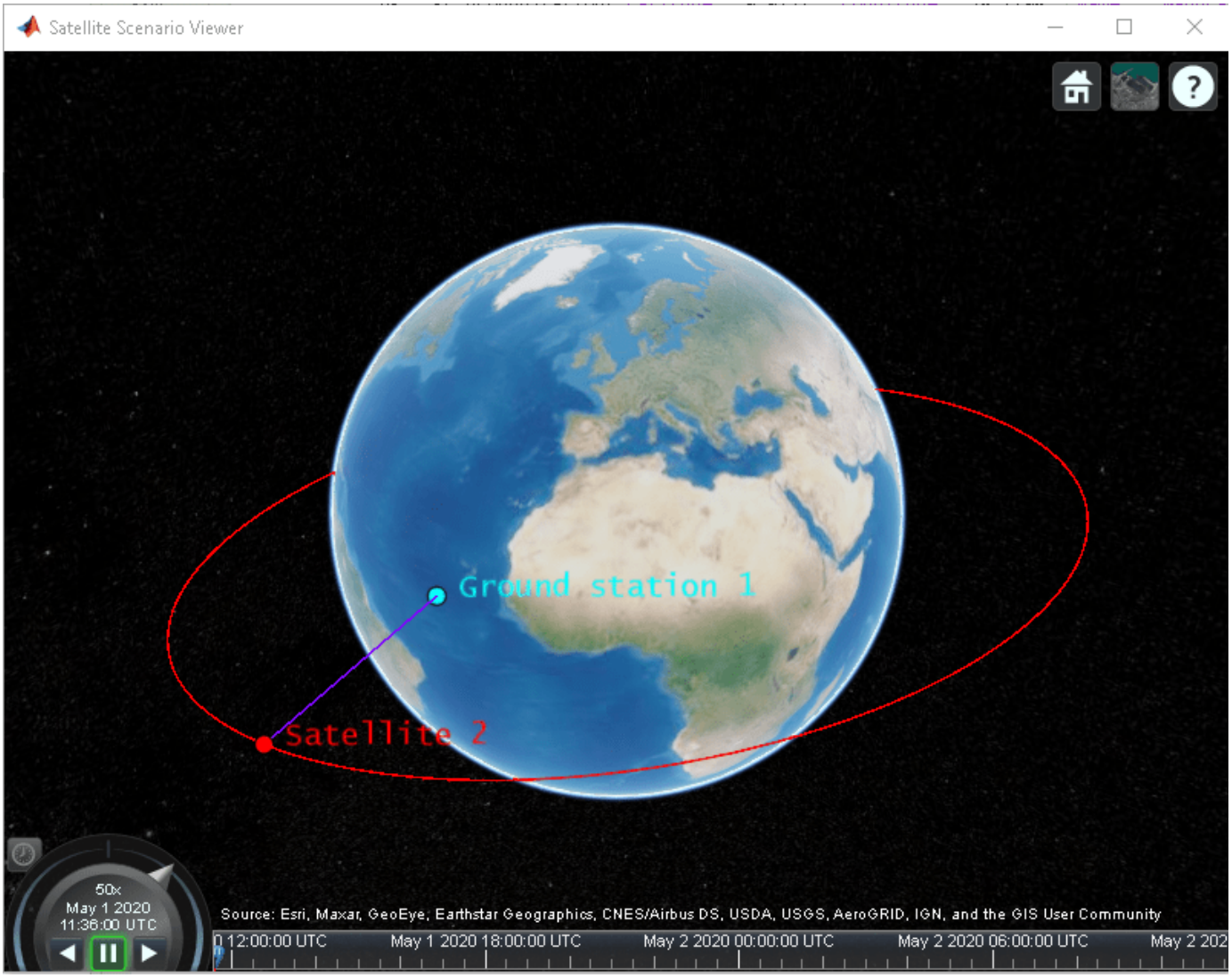
```
ac = access(sat, gs);
intvls = accessIntervals(ac)
```

```
intvls=8×8 table
       Source              Target          IntervalNumber         StartTime                EndTi
    _____________    ___________________    ______________    ____________________    ___________

    "Satellite 2"    "Ground station 1"          1           01-May-2020 11:36:00    01-May-2020
    "Satellite 2"    "Ground station 1"          2           01-May-2020 14:20:00    01-May-2020
    "Satellite 2"    "Ground station 1"          3           01-May-2020 17:27:00    01-May-2020
    "Satellite 2"    "Ground station 1"          4           01-May-2020 20:34:00    01-May-2020
    "Satellite 2"    "Ground station 1"          5           01-May-2020 23:41:00    02-May-2020
    "Satellite 2"    "Ground station 1"          6           02-May-2020 02:50:00    02-May-2020
    "Satellite 2"    "Ground station 1"          7           02-May-2020 05:59:00    02-May-2020
    "Satellite 2"    "Ground station 1"          8           02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `satellite` | `show` | `transmitter`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

ConicalSensor

Conical sensor object belonging to satellite scenario

Description

`ConicalSensor` defines a conical sensor object belonging to a satellite scenario.

Creation

You can create the `ConicalSensor` object using the `conicalSensor` object function of the `Satellite` or `GroundStation` objects.

Properties

`Name` — ConicalSensor name
`"ConicalSensor idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling ConicalSensor. After you call ConicalSensor, this property is read-only.

ConicalSensor name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one ConicalSensor is added, specify `Name` as a string scalar or a character vector.
- If multiple ConicalSensors are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the ConicalSensor added by the `ConicalSensor` object function. If another ConicalSensor of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`ID` — ConicalSensor ID assigned by simulator
real positive scalar

This property is set internally by the simulator and is read-only.

ConicalSensor ID assigned by the simulator, specified as a positive scalar.

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting location with respect to the parent object, specified as a three-element row vector of positive numbers in meters. The position vector is specified in the body frame of the input `parent`.

`MaxViewAngle` — Field of view angle

`30` (default) | scalar in the range [0, 180]

Field of view angle, specified as a scalar in the range [0, 180]. Units are in degrees.

`Accesses` — Access analysis objects

row vector of `Access` objects

You can set this property only when calling ConicalSensor. After you call ConicalSensor, this property is read-only.

Access analysis objects, specified as a row vector of `Access` objects.

`FieldOfView` — Field of view objects

row vector of `FieldOfView` objects

You can set this property only when calling ConicalSensor. After you call ConicalSensor, this property is read-only.

Field of view objects, specified as a scalar of `FieldOfView` objects.

Object Functions

access	Add access analysis objects to satellite scenario
fieldOfView	Visualize field of view of conical sensor

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`access` | `groundStation` | `hide` | `play` | `receiver` | `show` | `transmitter`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

Transmitter

Transmitter object belonging to satellite scenario

Description

Transmitter defines a transmitter object belonging to a satellite scenario.

Creation

You can create Transmitter objects using the `transmitter` method of `satellite`, `groundStation`, or `gimbal`.

Properties

`Name` — Transmitter name
`"Transmitter idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling Transmitter. After you call Transmitter, this property is read-only.

Transmitter name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one Transmitter is added, specify `Name` as a string scalar or a character vector.
- If multiple Transmitters are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the Transmitter added by the `Transmitter` object function. If another Transmitter of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`ID` — Transmitter ID assigned by simulator
real positive scalar

This property is set internally by the simulator and is read-only.

Transmitter ID assigned by the simulator, specified as a positive scalar.

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting location with respect to the parent object, specified as a three-element row vector of positive numbers in meters. The position vector is specified in the body frame of the input `parent`.

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting orientation with respect to parent object, specified as a three-element row vector of positive numbers in degrees. The elements of the vector correspond to yaw, pitch, and roll in that order. Yaw, pitch, and roll are positive rotations about the parent's z - axis, intermediate y - axis and intermediate x - axis of the parent.

Example: `[0; 30; 60]`

`Antenna` — Antenna object associated with Transmitter
`gaussianAntenna` object

Antenna object associated with the Transmitter, specified as an antenna object. The default gaussian antenna has a dish diameter of 1 m and an aperture efficiency of 0.65.

`SystemLoss` — Total loss in Transmitter
`5` (default) | positive scalar

Total loss in the Transmitter, specified as a real positive scalar. Units are in dB.

`Frequency` — Transmitter frequency
`14e9` (default) | positive scalar

Transmitter frequency, specified as a positive scalar. Units are in Hz.

`BitRate` — Bit rate of transmitter
`10` (default) | real positive scalar

Bit rate of the transmitter, specified as a real positive scalar. Units are in Mbps.

`Power` — Power of high power amplifier
`12` (default) | real positive scalar

Power of the high power amplifier, specified as a real positive scalar. Units are in dbW.

`Links` — Link analysis objects
row vector of `Link` objects

You can set this property when calling Transmitter only. After you call Transmitter, this property is read-only.

Link analysis objects, specified as a row vector `Link` objects.

Object Functions

gaussianAntenna	Add Gaussian antennas
link	Add link analysis objects to transmitter

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `groundStation` | `hide` | `play` | `show`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Multi-Hop Satellite Communications Link Between Two Ground Stations”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

Receiver

Receiver object belonging to satellite scenario

Description

The `Receiver` object defines a receiver object function belonging to the satellite scenario.

Creation

You can create `Receiver` object using the `receiver` object function of the `Satellite`, `GroundStation`, or `Gimbal` object.

Properties

`Name` — Receiver name

`"Receiver idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling Receiver. After you call Receiver, this property is read-only.

Receiver name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one Receiver is added, specify `Name` as a string scalar or a character vector.
- If multiple Receivers are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the Receiver added by the `Receiver` object function. If another Receiver of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`ID` — Receiver ID assigned by simulator

real positive scalar

This property is set internally by the simulator and is read-only.

Receiver ID assigned by the simulator, specified as a positive scalar.

`MountingLocation` — Mounting location with respect to parent

`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting location with respect to the parent object, specified as a three-element row vector of positive numbers in meters. The position vector is specified in the body frame of the input `parent`.

`MountingAngles` — Mounting orientation with respect to parent object

`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting orientation with respect to parent object, specified as a three-element row vector of positive numbers in degrees. The elements of the vector correspond to yaw, pitch, and roll in that order. Yaw, pitch, and roll are positive rotations about the parent's z - axis, intermediate y - axis and intermediate x - axis of the parent.

Example: `[0; 30; 60]`

`Antenna` — Antenna object associated with Receiver

`gaussianAntenna` object

Antenna object associated with the Receiver, specified as an antenna object. The default gaussian antenna has a dish diameter of 1 m and an aperture efficiency of 0.65.

`SystemLoss` — Total loss in Receiver

5 (default) | positive scalar

Total loss in the Receiver, specified as a real positive scalar. Units are in dB.

`GainToNoiseTemperatureRatio` — Gain to noise temperature ratio

3 (default) | scalar

Gain to noise temperature ratio of the antenna, specified as the comma-separated pair consisting of `'GainToNoiseTemperatureRatio'` and a scalar. Units are in dB/K.

`RequiredEbNo` — Lowest Eb/No necessary for link closure

`10` (default) | positive scalar

Lowest energy per bit to noise power spectral density ratio (Eb/No) necessary for link closure, specified as the comma-separated pair consisting of `'RequiredEbNo'` and a positive scalar. Units are in dB.

Object Functions

gaussianAntenna Add Gaussian antennas

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`access` | `groundStation` | `hide` | `link` | `play` | `show`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Multi-Hop Satellite Communications Link Between Two Ground Stations”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

Gimbal

Gimbal object belonging to satellite scenario

Description

The `Gimbal` defines a gimbal object belonging to a satellite scenario.

Creation

You can create a `Gimbal` object using the `gimbal` object function of the `Satellite` or `GroundStation`.

Properties

`Name` — Gimbal name
`"Gimbal idx"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling Gimbal. After you call Gimbal, this property is read-only.

Gimbal name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one Gimbal is added, specify `Name` as a string scalar or a character vector.
- If multiple Gimbals are added, specify `Name` as a string vector or a cell array of character vectors. The number of elements in the string vector or cell array must be equal to the number of satellites being added.

In the default value, *idx* is the count of the Gimbal added by the `Gimbal` object function. If another Gimbal of the same name exists, a suffix $\_idx_2$ is added, where idx_2 is an integer that is incremented by 1 starting from 1 until the name duplication is resolved.

Data Types: `char` | `string`

`ID` — Gimbal ID assigned by simulator
real positive scalar

This property is set internally by the simulator and is read-only.

Gimbal ID assigned by the simulator, specified as a positive scalar.

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting location with respect to the parent object, specified as a three-element row vector of positive numbers in meters. The position vector is specified in the body frame of the input `parent`.

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers

Mounting orientation with respect to parent object, specified as a three-element row vector of positive numbers in degrees. The elements of the vector correspond to yaw, pitch, and roll in that order. Yaw, pitch, and roll are positive rotations about the parent's z - axis, intermediate y - axis and intermediate x - axis of the parent.

Example: `[0; 30; 60]`

`ConicalSensors` — Conical sensors
row vector of conical sensors

You can set this property only when calling `conicalSensor`. After you call `conicalSensor`, this property is read-only.

Conical sensors attached to the Gimbal, specified as a row vector of conical sensors.

`Receivers` — Receivers attached to Gimbal
row vector of `Receiver` objects

You can set this property only when calling `receiver`. After you call `receiver`, this property is read-only.

Receivers attached to the Gimbal, specified as a row vector of `Receiver` objects.

`Transmitters` — Transmitters attached to Gimbal
row vector of `Transmitter` objects

You can set this property only when calling `transmitter`. After you call `transmitter`, this property is read-only.

Transmitters attached to the Gimbal, specified as a row vector of `Transmitter` objects.

Object Functions

transmitter	Add transmitter to satellite scenario
receiver	Add receiver to satellite scenario
conicalSensor	Add conical sensor to satellite scenario
pointAt	Target at which entity must be pointed
gimbalAngles	Steering angles of gimbal

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `conicalSensor` | `groundStation` | `hide` | `play` | `receiver` | `satellite` | `show` | `transmitter`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

FieldOfView

Field of view object belonging to satellite scenario

Description

The FieldOfView object defines a field of view object belonging to a satellite scenario.

Creation

You can create a `FieldOfView` object using the `fieldOfView` object function of the `ConicalSensor` object.

Properties

`LineWidth` — Visual width of field of view contour
1 (default) | scalar in the range (0 10]

Visual width of the field of view contour in pixels, specified as a scalar in the range (0 10].

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LineColor` — Color of field of view contour
`[0 1 0]` (default) | RGB triplet | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of field of view contour, specified as an RGB triplet, hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'cyan'	'c'	[0 1 1]	'#00FFFF'	
'magenta'	'm'	[1 0 1]	'#FF00FF'	
'yellow'	'y'	[1 1 0]	'#FFFF00'	
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	
'none'	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

Example: 'blue'

Example: [0 0 1]

Example: '#0000FF'

VisibilityMode — Visibility mode of field of view contour
'inherit' (default) | 'manual'

Visibility mode of the field of view contour, specified as one of these values:

- 'inherit' — Visibility of the graphic matches that of the parent
- 'manual' — Visibility of the graphic is not inherited and is independent of that of the parent

Object Functions

show Show object in satellite scenario viewer
hide Hides satellite scenario entity from viewer

See Also

Objects
satelliteScenario | satelliteScenarioViewer

Functions
access | groundStation | hide | play | show

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

Link

Link analysis object belonging to Transmitter

Description

The `Link` object defines a link analysis object belonging to `Transmitter`.

Creation

You can create a `Link` object using the `link` object function of the `Transmitter` or `Receiver` objects.

Properties

`Sequence` — Transmitter or receiver ID
vector of positive numbers

You can set this property only when calling Link. After you call Link, this property is read-only.

Transmitter or receiver ID, specified as a vector of positive numbers.

`LineWidth` — Visual width of link line
1 (default) | scalar in the range (0 10]

Visual width of link line in pixels, specified as a scalar in the range (0 10].

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LineColor` — Color of link line
`[0 1 0]` (default) | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the link line, specified as an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'red'	'r'	[1 0 0]	'#FF0000'	
'green'	'g'	[0 1 0]	'#00FF00'	
'blue'	'b'	[0 0 1]	'#0000FF'	
'cyan'	'c'	[0 1 1]	'#00FFFF'	
'magenta'	'm'	[1 0 1]	'#FF00FF'	
'yellow'	'y'	[1 1 0]	'#FFFF00'	
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	
'none'	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

Object Functions

show	Show object in satellite scenario viewer
ebno	Eb/No at final node of link
linkPercentage	Percentage of time when link between first and last node in link analysis is closed
linkStatus	Status of link closure between first and last node
linkIntervals	Intervals during which link is closed
hide	Hides satellite scenario entity from viewer

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`groundStation` | `hide` | `play` | `receiver` | `show` | `transmitter`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

GroundTrack

Ground track object belonging to satellite in scenario

Description

The `GroundTrack` object defines a ground track object belonging to a satellite in a scenario.

Creation

You can create a `GroundTrack` object using the `groundTrack` object function of the `Satellite` object.

Properties

`LeadTime` — Period of ground track to be visualized
`StartTime` to `StopTime` (default) | positive scalar

Period of the ground track to be visualized in the satellite scenario viewer, specified as a comma-separated pair consisting of `'LeadTime'` and a real positive scalar in seconds.

`TrailTime` — Period of ground track history to be visualized
`StartTime` to `StopTime` (default) | positive scalar

Period of the ground track history to be visualized in `Viewer`, specified as a comma-separated pair consisting of `'TrailTime'` and a real positive scalar in seconds.

`LineWidth` — Visual width of ground track
`1` (default) | scalar in the range (0 10]

Visual width of the ground track in pixels, specified as a comma-separated pair consisting of `'LineWidth'` and a scalar in the range (0 10].

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LeadLineColor` — Color of future ground track line
`[1 0 1]` (default) | RGB triplet | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the future ground track line, specified as a comma-separated pair consisting of `'LeadLineColor'` and an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.

- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	
`[0.8500 0.3250 0.0980]`	`'#D95319'`	
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

`TrailLineColor` — Color of ground track line history

`[1 0.5 0]` (default) | RGB triplet | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the ground track line history, specified as a comma-separated pair consisting of `'TrailLineColor'` and an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	
`[0.8500 0.3250 0.0980]`	`'#D95319'`	
`[0.9290 0.6940 0.1250]`	`'#EDB120'`	
`[0.4940 0.1840 0.5560]`	`'#7E2F8E'`	
`[0.4660 0.6740 0.1880]`	`'#77AC30'`	
`[0.3010 0.7450 0.9330]`	`'#4DBEEE'`	
`[0.6350 0.0780 0.1840]`	`'#A2142F'`	

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

`VisibilityMode` — Visibility mode of ground track

`'inherit'` (default) | `'manual'`

Visibility mode of the ground track, specified as either one of these values:

- `'inherit'` — Visibility of the graphic matches that of the parent

- `'manual'` — Visibility of the graphic is not inherited and is independent of that of the parent

Object Functions

show Show object in satellite scenario viewer
hide

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`access` | `groundStation` | `hide` | `play` | `satellite` | `show`

Topics

“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Introduced in R2021a

System Objects

dvbs2WaveformGenerator

Generate DVB-S2 waveform

Description

The `dvbs2WaveformGenerator` System object generates a Digital Video Broadcasting Satellite Second Generation (DVB-S2) time-domain waveform consisting of a single or multiple physical layer frames. The object implements the waveform generation aspects of ETSI EN 302 307-1 V1.4.1 (2014-11) [1].

To generate a DVB-S2 waveform:

1. Create the `dvbs2WaveformGenerator` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?.

Creation

Syntax

```
s2waveGen = dvbs2WaveformGenerator
s2waveGen = dvbs2WaveformGenerator(Name,Value)
```

Description

`s2waveGen = dvbs2WaveformGenerator` creates a default DVB-S2 waveform generator System object.

`s2waveGen = dvbs2WaveformGenerator(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `dvbs2WaveformGenerator('NumInputStreams',4,'UPL',100)` specifies four input streams, each with a user packet length of 100 bits.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`StreamFormat` — Input stream format

`"TS"` (default) | `"GS"`

Input stream format, specified as one of these values.

- `"TS"` — For transport stream format
- `"GS"` — For generic stream format

Data Types: `char` | `string`

`NumInputStreams` — Number of input streams

`1` (default) | integer in the range [1, 256]

Number of input streams, specified as an integer in the range [1, 256].

Data Types: `double`

`UPL` — User packet length

`0` (default) | nonnegative integer | vector of nonnegative integers

User packet (UP) length in bits, specified as one of these options.

- Nonnegative integer — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, the UP in each stream must be equal to the integer value of the `UPL` property.
- Vector of nonnegative integers — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the UP in each stream must be the size of the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

> **Note** When you specify `UPL` as a multi-input stream, all UPs must be either packetized or in a continuous stream. Mixing stream types is not supported.

The maximum value of `UPL` as an integer scalar or an integer element in the row vector must be less than or equal to the corresponding `DFL` property value.

For a generic continuous stream, set `UPL` to `0`.

Dependencies

To enable this property, set the `StreamFormat` property to `"GS"`. If you set the `StreamFormat` property to `"TS"`, the UPL is fixed to 1504 bits.

Data Types: `double`

`FECFrame` — FEC frame format

`"normal"` (default) | `"short"`

Forward error correction (FEC) frame format, specified as one of these two options.

- `"normal"` — Sets the low density parity-check (LDPC) codeword length to 64,800 bits
- `"short"` — Sets the LDPC codeword length to 16,200 bits

Tunable: Yes

Data Types: `char` | `string`

`MODCOD` — Modulation scheme and FEC rate

`1` (default) | integer in the range [1, 28] | vector of integers in the range [1, 28]

Modulation scheme and FEC rate for input transmission, specified as one of these options, as defined in ETSI EN 302 307-1 Section 5.5.2.2 Table 12 [1].

- Integer in the range [1, 28] — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, each stream has the same modulation scheme and coding rate.
- Vector of integers in the range [1, 28] — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, each input stream has a modulation scheme and coding rate equal to the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

Note `MODCOD` values `11`, `17`, `23`, and `28` are not valid when you set the `FECFrame` property to `"short"` (as specified in ETSI EN 302 307-1 Section 5.3 Table 5b [1]).

Tunable: Yes

Data Types: `double`

`DFL` — Data field length

`15,928` (default) | integer in the range [1, (K_{BCH}−80)] | vector of integers in the range [1, (K_{BCH}−80)]

Data field (DF) length in bits, specified as one of these options. K_{BCH} is the uncoded BCH block length, as specified in ETSI EN 302 307-1 Section 5.3 Table 5a and 5b [1].

- Integer in the range [1, (K_{BCH}−80)] — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, the length of the DF in baseband frame of each stream is the same value.
- Vector of integers in the range [1, (K_{BCH}−80)] — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the length of the data field in the baseband frame of each stream must be the size of the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

Tunable: Yes

Data Types: `double`

`ScalingMethod` — Constellation amplitude scaling method

`"outer radius as 1"` (default) | `"unit average power"`

Constellation amplitude scaling method, specified as `"outer radius as 1"` or `"unit average power"`.

Dependencies

To enable this property, set the `MODCOD` property to a value in the range [18, 28], which indicates only 16APSK and 32APSK modulation schemes.

Data Types: `char` | `string`

`HasPilots` — Pilot block indication

`0` or `false` (default) | `1` or `true` | vector of `logical` values

Pilot block indication, specified as a logical value of `0` (`false`), `1` (`true`), or a vector of `logical` values. Set this value to `1` (`true`) to indicate pilots are inserted in the physical layer frame.

If you set the `NumInputStreams` property to a value greater than 1, you can configure pilots for each stream by specifying this property as a vector. The length of this vector must be equal to `NumInputStreams`.

Tunable: Yes

Data Types: `logical`

`RolloffFactor` — Transmit filter roll-off factor

`0.35` (default) | `0.25` | `0.2`

Transmit filter roll-off factor for baseband pulse shaping, specified as `0.35`, `0.25`, or `0.2`.

Data Types: `double`

`FilterSpanInSymbols` — Filter span in symbols

`10` (default) | positive integer

Filter span in symbols, specified as a positive integer.

The ideal impulse response of the transmit filter is truncated to a length that spans the number of symbols specified in this property.

Data Types: `double`

`SamplesPerSymbol` — Number of samples per symbol

`4` (default) | positive integer

Number of samples per symbol, specified as a positive integer.

Data Types: `double`

`ISSYI` — Input stream synchronization indicator

`0` or `false` (default) | `1` or `true`

Input stream synchronization (ISSY) indicator, specified as a logical value of `0` (`false`) or `1` (`true`). To indicate that input stream synchronization is used, set this value to `1` (`true`).

Dependencies

To enable this property, set the `NumInputStreams` property to a value greater than 1 and set the `UPL` property to a nonzero value.

Data Types: `logical`

`ISCRFormat` — Input stream clock reference format

`"short"` (default) | `"long"`

Input stream clock reference format, specified as one of these options.

- `"short"` — Indicates the length of ISSY as 2 bytes
- `"long"` — Indicates the length of ISSY as 3 bytes

When you set the `StreamFormat` property to `"GS"`, `NumInputStreams` property to a value greater than 1, `UPL` property to a nonzero value, and `ISSYI` to `1` (`true`), only the `"short"` option of this `ISCRFormat` property is applicable.

Dependencies

To enable this property, set the `StreamFormat` property to `"TS"`, the `NumInputStreams` property to a value greater than 1, and the `ISSYI` property to `1` (`true`).

Data Types: `char` | `string`

`MinNumPackets` — Minimum number of packets required to create DF

integer in the range [1, 58,112] | row vector of integers

This property is read-only.

Minimum number of packets required to create a DF, specified as one of these options.

- Integer in the range [1, 58,112] — This option applies with single-input streams only.
- Row vector of integers in the range [1, 58,112] — This option applies with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the minimum number of packets required for each stream is equal to the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

The value of `MinNumPackets` is computed based of values of `DFL` and `UPL` properties.

Dependencies

To enable this property, set the `UPL` property to a nonzero value.

Data Types: `double`

Usage

Syntax

```
txWaveform = s2waveGen(data)
```

Description

`txWaveform = s2waveGen(data)` generates a DVB-S2 time-domain waveform from the input information bits.

Input Arguments

`data` — Input information bits

binary-valued column vector | cell array of binary-valued column vectors

Input information bits, specified as one of these options. Each element of the column vector or cell array can be of data type `double`, `int8`, or `logical`.

- Binary-valued column vector — Use this option with single-input streams.
- Cell array of binary-valued column vectors — Use this option with multi-input streams. Each element of the array represents the corresponding input stream. The length of the cell array must be equal to the value of the `NumInputStreams` property.

Input `data`, either as a single-input or multi-input stream, must be input in one of these forms.

- Packetized stream — The number of packets in each stream must be an integer multiple of the `MinNumPackets` property.

  For a packetized stream, an 8-bit sync field must be included at the beginning of each packet. The combined length of a packet and its sync bits must be equal to the corresponding value of the `UPL` property.
- Continuous stream — The number of bits for each stream must be an integer multiple of the `DFL` property.

Note When you set the `StreamFormat` property to `"TS"`, the sync byte is fixed as 47 hex.

Data Types: `double` | `int8` | `logical` | `cell`

Output Arguments

`txWaveform` — Generated time-domain DVB-S2 waveform
column vector

Generated time-domain DVB-S2 waveform, returned as a column vector. The waveform is generated in the form of complex in-phase quadrature (IQ) samples and can consist of a single physical layer frame or multiple physical layer frames.

When you set the `NumInputStreams` property to a value greater than 1, the data fields generated from different input streams are merged using the round-robin technique.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `dvbs2WaveformGenerator`

info	Characteristic information about object
flushFilter	Flush transmit filter

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object
isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Generate DVB-S2 Waveform for Single-Input Stream

Generate a Digital Video Broadcasting Satellite Second Generation (DVB-S2) time-domain waveform for a single-input transport stream (TS) with a single physical layer (PL) frame per stream. Visualize the waveform using constellation plots and signal spectrum.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream.

```
numFrames = 1;
```

Create a DVB-S2 System object. Specify the modulation scheme and FEC rate (MODCOD) and data field length (DFL).

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.MODCOD = 21;              % 16APSK 5/6
s2WaveGen.DFL = 39690;
s2WaveGen.HasPilots = true;         % Pilot insertion indication
disp(s2WaveGen)
```

```
  dvbs2WaveformGenerator with properties:

          StreamFormat: "TS"
       NumInputStreams: 1
             FECFrame: "normal"
               MODCOD: 21
                  DFL: 39690
         ScalingMethod: "outer radius as 1"
             HasPilots: 1
         RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 4

  Show all properties
```

Create a bit vector of information bits, `data`, of concatenated TS user packets.

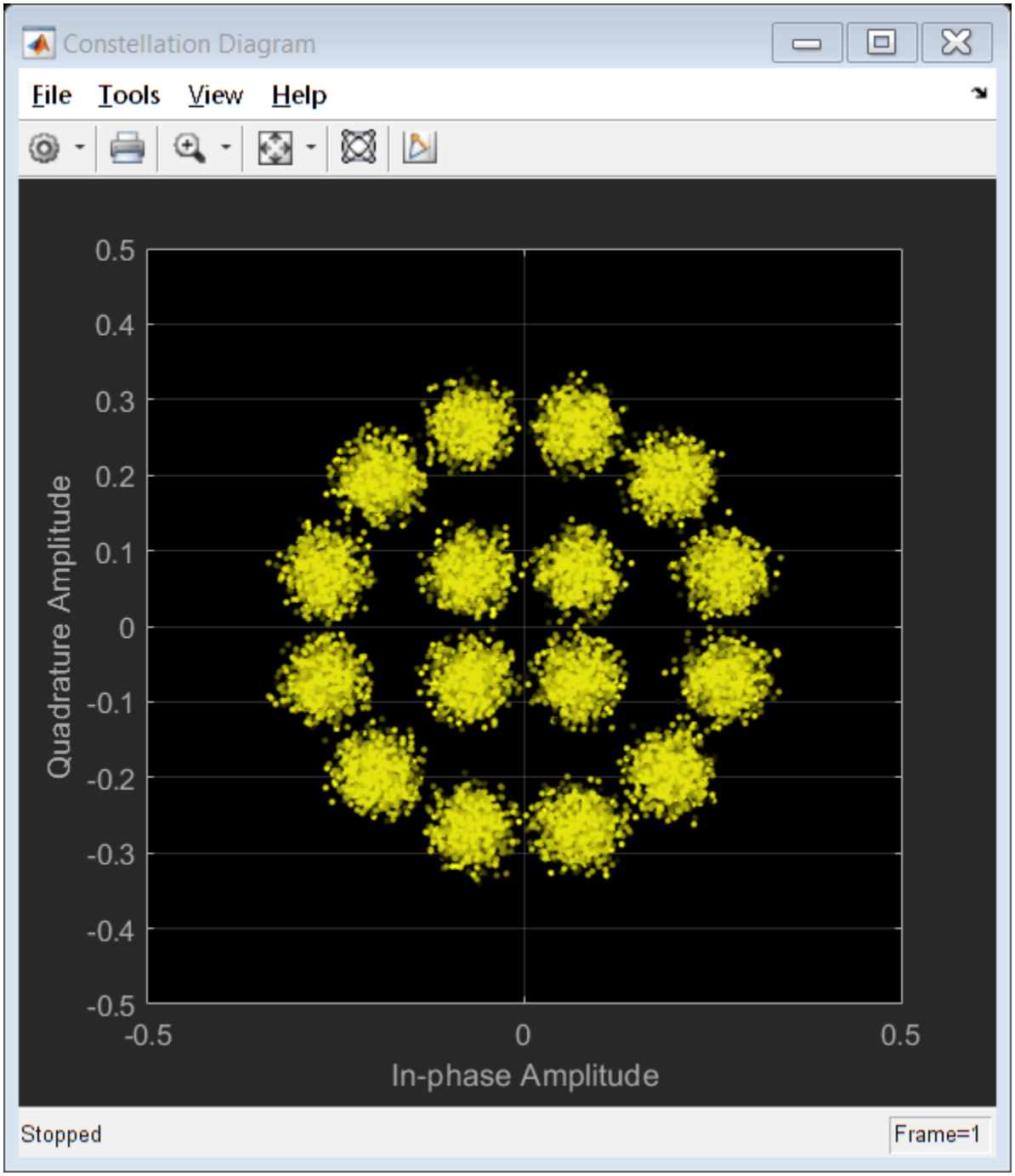
```
syncBits = [0 1 0 0 0 1 1 1]';    % Sync byte for TS packet is 47 Hex
pktLen = 1496;                    % UP length without sync bits is 1496
numPkts = s2WaveGen.MinNumPackets*numFrames;
txRawPkts = randi([0 1],pktLen,numPkts);
txPkts = [repmat(syncBits,1,numPkts); txRawPkts];
data = txPkts(:);
```

Generate a DVB-S2 time-domain waveform using the information bits, `data`.

```
txWaveform = s2WaveGen(data);
```

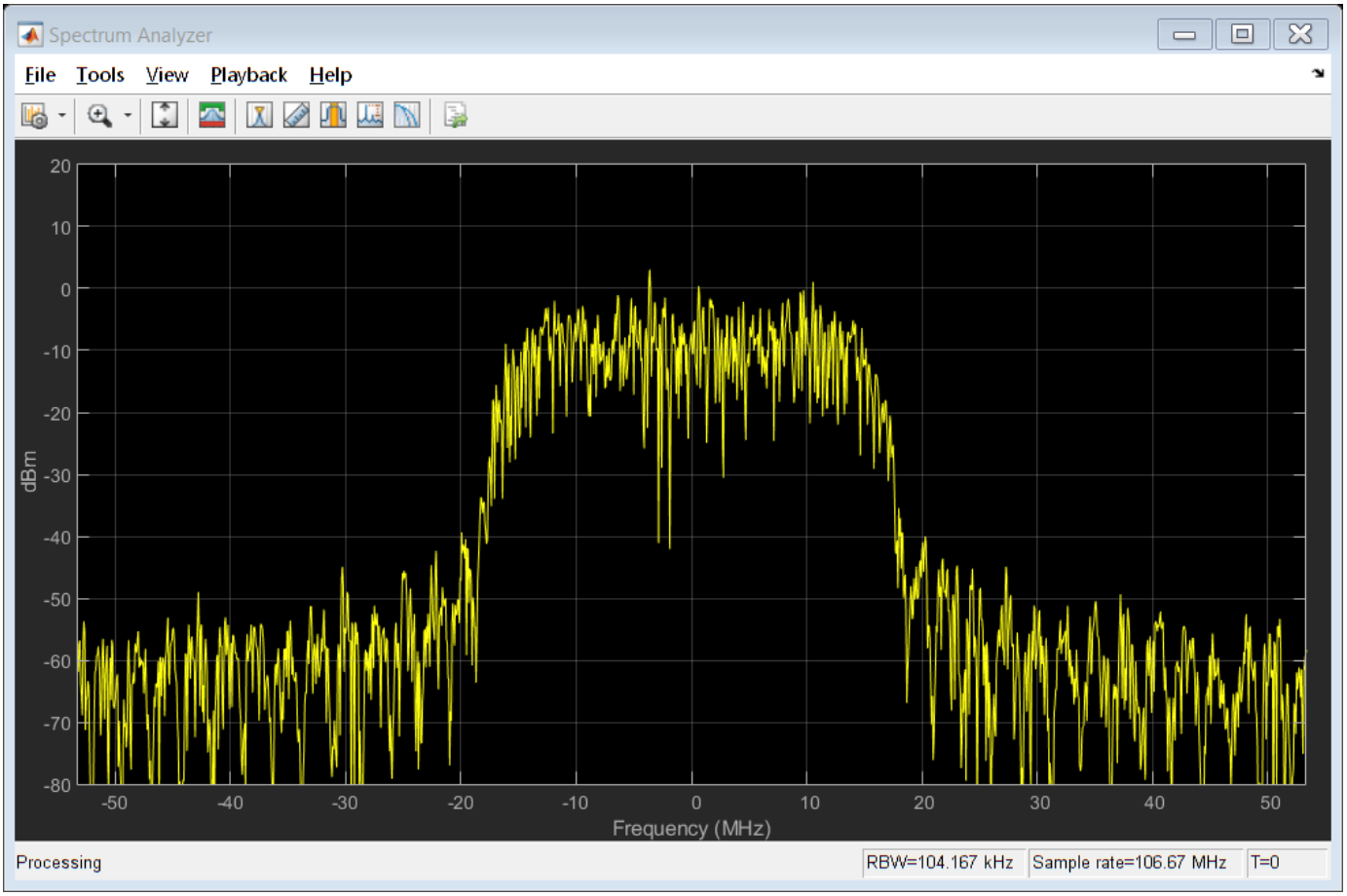
Visualize the constellation plot for the generated DVB-S2 time-domain waveform by creating a comm.ConstellationDiagram System object.

```
sps = s2WaveGen.SamplesPerSymbol;
constel = comm.ConstellationDiagram('ColorFading',true, ...
    'ShowTrajectory',0, ...
    'SamplesPerSymbol',sps, ...
    'ShowReferenceConstellation',false, ...
    'XLimits',[-0.5 0.5], 'YLimits',[-0.5 0.5]);
plHeaderLen = 90*sps;           % PL header length
constel(txWaveform(plHeaderLen+1:end));
release(constel);
```



Display the frequency spectrum of the generated DVB-S2 time-domain waveform by creating a `dsp.SpectrumAnalyzer` System object.

```
BW = 36e6;                  % Typical satellite channel bandwidth
Fsym = BW/(1+s2WaveGen.RolloffFactor);
Fsamp = Fsym*sps;
scope = dsp.SpectrumAnalyzer('SampleRate',Fsamp);
scope(txWaveform)
```



Generate DVB-S2 Waveform for Multi-Input Stream

Generate a Digital Video Broadcasting Satellite Second Generation (DVB-S2) time-domain waveform for a multi-input generic stream (GS) with multiple physical layer (PL) frames per stream.

This example requires MAT-files with LDPC parity matrices. If they are not available on the path, execute the following commands to download and unzip the MAT-files.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip', 'file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream.

```
numFrames = 3;
```

Create a DVB-S2 System object with variable coding and modulation configuration for a multi-input GS. Specify the modulation scheme and FEC rate (MODCOD) and data field length (DFL).

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.StreamFormat = "GS";
s2WaveGen.NumInputStreams = 2;
s2WaveGen.MODCOD = [6 24];          % QPSK 2/3 and 32APSK 3/4
s2WaveGen.DFL = [42960 48328];
s2WaveGen.HasPilots = true;
s2WaveGen.SamplesPerSymbol = 10;
disp(s2WaveGen)
```

```
  dvbs2WaveformGenerator with properties:

          StreamFormat: "GS"
       NumInputStreams: 2
                   UPL: 0
             FECFrame: "normal"
               MODCOD: [6 24]
                  DFL: [42960 48328]
         ScalingMethod: "outer radius as 1"
             HasPilots: 1
         RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 10
```

Create a bit vector of input information bits for each GS user packet.

```
data = cell(s2WaveGen.NumInputStreams,1);
   for i = 1:s2WaveGen.NumInputStreams
           data{i} = randi([0 1],s2WaveGen.DFL(i)*numFrames,1,'int8');
   end
```

Generate the DVB-S2 time-domain waveform with the input information bits.

```
txWaveform = s2WaveGen(data);
```

References

[1] ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- For all properties that support string and cell array input, code generation is only supported with cell array input.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Functions
`dvbs2BitRecover`

Objects
`dvbs2xWaveformGenerator`

Introduced in R2021a

dvbs2xWaveformGenerator

Generate DVB-S2X waveform

Description

The `dvbs2xWaveformGenerator` System object generates a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) time-domain waveform consisting of a single or multiple physical layer (PL) frames. The object implements the waveform generation aspects of ETSI EN 302 307-2 V1.1.1 (2015-11) [2].

To generate a DVB-S2X waveform:

1. Create the `dvbs2xWaveformGenerator` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?.

Creation

Syntax

```
s2xWaveGen = dvbs2xWaveformGenerator
s2xWaveGen = dvbs2xWaveformGenerator(Name,Value)
```

Description

`s2xWaveGen = dvbs2xWaveformGenerator` creates a default DVB-S2X waveform generator System object.

`s2xWaveGen = dvbs2xWaveformGenerator(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `dvbs2xWaveformGenerator('NumInputStreams',4,'UPL',100)` specifies four input streams, each with a user packet length of 100 bits.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`StreamFormat` — Input stream format

`"TS"` (default) | `"GS"`

Input stream format, specified as one of these values.

- `"TS"` — For transport stream format
- `"GS"` — For generic stream format

Data Types: `char` | `string`

`HasTimeSlicing` — Time slicing indicator
`0` or `false` (default) | `1` or `true`

Time slicing indicator, specified as a logical value of `0` (`false`) or `1` (`true`). To indicate that time slicing transmission format is used, set this value to `1` (`true`).

If you set this property to `1` (`true`), you can set the `NumInputStreams` property to a maximum value of `8`.

Data Types: `logical`

`NumInputStreams` — Number of input streams
`1` (default) | integer in the range [1, 256]

Number of input streams, specified as an integer in the range [1, 256].

When you set the `HasTImeSlicing` property to `true`, `NumInputStreams` property can be specified to a maximum value of `8`.

Data Types: `double`

`UPL` — User packet length
`0` (default) | nonnegative integer | vector of nonnegative integers

User packet (UP) length in bits, specified as one of these options.

- Nonnegative integer — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, the UP in each stream must be equal to the integer value of the `UPL` property.
- Vector of nonnegative integers — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the UP in each stream must be the size of the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

> **Note** When you specify `UPL` as a multi-input stream, all UPs must be either packetized or in a continuous stream. Mixing stream types is not supported.

The maximum value of `UPL` as an integer scalar or an integer element in the row vector must be less than or equal to the corresponding `DFL` property value.

For a generic continuous stream, set `UPL` to `0`.

Dependencies

To enable this property, set the `StreamFormat` property to `"GS"`. If you set the `StreamFormat` property to `"TS"`, the UPL is fixed to 1504 bits.

Data Types: `double`

`PLSDecimalCode` — PL signalling code information
`132` (default) | integer in the range [4, 249] | vector of integers in the range [4, 249]

PL signalling code information, in decimal format, specified as one of these options (as described in ETSI EN 302 307-1 Section 5.5.2.2 [1] and ETSI EN 302 307-2 Section 5.5.2.2 Table 17a [2]).

- Integer in the range [4, 249] — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, each stream has the same modulation scheme and coding rate.
- Vector of integers in the range [4, 249] — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, each stream has a modulation scheme and coding rate equal to the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

All odd integer values are considered to include pilots in PL frames.

Note Few `PLSDecimalCode` values are invalid in this specified value range. Invalid values include {46, 47, 70, 71, 94, 95, 114, 128, 130, 176, 177, 188, 189, 192, 193, 196, and 197}.

To calculate the `PLSDecimalCode` property value for a DVB-S2X system configuration, use this formula.

`MODCOD`*4 + (0 - for normal `FECFrame`/1 - for short `FECFrame`)*2 + (0 - if `HasPilots` property value is `false`/1 - if `HasPilots` property value is `true`)

For example, if `MODCOD` = 18 (16APSK 2/3) with short FEC frame and pilots disabled, the value of `PLSDecimalCode` calculated by using this formula is:

`PLSDecimalCode` = 18*4 + 1*2 + 0 = 74

Note For very low signal to noise ratio (VL-SNR) frames, you must set the `PLSDecimalCode` property to either `129` or `131`, which indicates the VL-SNR set 1 or 2, respectively.

VL-SNR frames must not be combined with regular frames.

Tunable: Yes

Data Types: `double`

`CanonicalMODCODName` — Canonical modulation scheme and code rate name

`"QPSK 2/9"` (default) | character vector | string scalar | cell array | string array

Canonical modulation scheme and code rate name for VL-SNR frame transmission, specified as one of these options (as described in ETSI EN 302 307-2 Section 5.5.2.2 Table 18a [2]).

- Character vector or string scalar — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, each stream has the same modulation scheme and coding rate.
- Cell array or string array — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, each input stream has a modulation scheme and coding rate equal to the corresponding value in this array. The length of this array must be equal to `NumInputStreams`.

Valid `CanonicalMODCODName` values include these options.

- `"QPSK 2/9"`, `"BPSK 1/5"`, `"BPSK 11/45"`, `"BPSK-S 1/5"`, `"BPSK-S 11/45"`, and `"BPSK 1/3"` — Applicable for VL-SNR set 1
- `"BPSK 1/5"`, `"BPSK 4/15"`, and `"BPSK 1/3"` — Applicable for VL-SNR set 2

Tunable: Yes

Dependencies

To enable this property, set the `PLSDecimalCode` property to either `129` (for VL-SNR set 1) or `131` (for VL-SNR set 2). This property applies for only VL-SNR frame transmissions.

Data Types: `char` | `string`

`DFL` — Data field length

`18,448` (default) | integer in the range [1, (K_{BCH}−80)] | vector of integers in the range [1, (K_{BCH}−80)]

Data field (DF) length in bits, specified as one of these options. K_{BCH} is the uncoded BCH block length, as specified in ETSI EN 302 307-1 Section 5.3 Table 5a and 5b [1].

- Integer in the range [1, (K_{BCH}−80)] — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, the length of the DF in baseband frame of each stream is the same value.
- Vector of integers in the range [1, (K_{BCH}−80)] — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the length of the data field in the baseband frame of each stream must be the size of the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

Tunable: Yes

Data Types: `double`

`ScalingMethod` — Constellation amplitude scaling method

`"outer radius as 1"` (default) | `"unit average power"`

Constellation amplitude scaling method, specified as `"outer radius as 1"` or `"unit average power"`.

Dependencies

To enable this property, set the `PLSDecimalCode` property to a value corresponding to APSK modulation, with the following as exception: {164, 165, 158, 159, 206, 207, 212, and 213}. The other exceptions are QPSK and 8 PSK values: {4 to 69, inclusive; 129; 131; 132 to 137, inclusive; 142 to 147, inclusive; 216 to 235, inclusive}.

Data Types: `char` | `string`

`PLScramblingIndex` — PL scrambling sequence index

integer in the range [0, 7] | vector of integers in the range [0, 7]

PL scrambling sequence index, specified as one of these options (as described in ETSI EN 302 307-2 Section 5.5.4 Table 19e [2]).

- Integer in the range [0, 7] — Use this option with single-input and multi-input streams.

  If you set the `NumInputStreams` property to a value greater than 1, each stream has the same value of PL scrambling index.

- Vector of integers in the range [0, 7] — Use this option when you set the `HasTimeSlicing` property to `true` for multi-input streams.

  If you set the `NumInputStreams` property to a value greater than 1, the PL scrambling index value of each stream must be equal to the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

To generate the PL scrambling sequence, the actual index used is `PLScramblingIndex`*10949.

Data Types: `double`

`RolloffFactor` — Transmit filter roll-off factor
`0.35` (default) | `0.05` | `0.1` | `0.15` | `0.2` | `0.25`

Transmit filter roll-off factor for baseband pulse shaping, specified as `0.35`, `0.05`, `0.1`, `0.15`, `0.2`, or `0.25`.

Data Types: `double`

`FilterSpanInSymbols` — Filter span in symbols
`10` (default) | positive integer

Filter span in symbols, specified as a positive integer.

The ideal impulse response of the transmit filter is truncated to a length that spans the number of symbols specified in this property.

Data Types: `double`

`SamplesPerSymbol` — Number of samples per symbol
`4` (default) | positive integer

Number of samples per symbol, specified as a positive integer.

Data Types: `double`

`ISSYI` — Input stream synchronization indicator
`0` or `false` (default) | `1` or `true`

Input stream synchronization (ISSY) indicator, specified as a logical value of `0` (`false`) or `1` (`true`). To indicate that input stream synchronization is used, set this value to `1` (`true`).

Dependencies

To enable this property, set the `NumInputStreams` property to a value greater than 1 and set the `UPL` property to a nonzero value.

Data Types: `logical`

`ISCRFormat` — Input stream clock reference format
`"short"` (default) | `"long"`

Input stream clock reference format, specified as one of these options.

- `"short"` — Indicates the length of ISSY as 2 bytes
- `"long"` — Indicates the length of ISSY as 3 bytes

When you set the `StreamFormat` property to `"GS"`, `NumInputStreams` property to a value greater than 1, `UPL` property to a nonzero value, and `ISSYI` to `1` (`true`), only the `"short"` option of this `ISCRFormat` property is applicable.

Dependencies

To enable this property, set the `StreamFormat` property to `"TS"`, the `NumInputStreams` property to a value greater than 1, and the `ISSYI` property to `1` (`true`).

Data Types: `char` | `string`

`MinNumPackets` — Minimum number of packets required to create DF
integer in the range [1, 58,112] | row vector of positive integers

This property is read-only.

Minimum number of packets required to create a DF, specified as one of these options.

- Integer in the range [1, 58,112] — This option applies with single-input streams only.
- Row vector of positive integers in the range [1, 58,112] — This option applies with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the minimum number of packets required for each stream is equal to the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

The value of `MinNumPackets` is computed based of values of `DFL` and `UPL` properties.

Dependencies

To enable this property, set the `UPL` property to a nonzero value.

Data Types: `double`

Usage

Syntax

```
txWaveform = s2xWaveGen(data)
```

Description

`txWaveform = s2xWaveGen(data)` generates a DVB-S2X time-domain waveform from the input information bits.

Input Arguments

`data` — Input information bits
binary-valued column vector | cell array of binary-valued column vectors

Input information bits, specified as one of these options. Each element of the column vector or cell array can be of the data type `double`, `int8`, or `logical`.

- Binary-valued column vector – Use this option with single-input stream.
- Cell array of binary-valued column vectors – Use this option with multi-input streams. Each element of the array represents the corresponding input stream. The length of the cell array must be equal to the value of the `NumInputStreams` property.

`data`, either single stream or multi-stream, can be input in one of these forms.

- Packetized stream – The number of packets in each stream must be an integer multiple of the `MinNumPackets` property.

  For a packetized stream, an 8-bit sync field must be included at the beginning of each packet. The combined length of a packet and its sync bits must be equal to the corresponding value of the `UPL` property.
- Continuous Stream – The number of bits for each stream must be an integer multiple of the `DFL` property.

Note When you set the `StreamFormat` property to `"TS"`, the sync byte is fixed as 47 hex.

Data Types: `double` | `int8` | `logical` | `cell`

Output Arguments

`txWaveform` — Generated time-domain DVB-S2X waveform
column vector

Generated time-domain DVB-S2X waveform, returned as a column vector. The waveform is generated in the form of complex in-phase quadrature (IQ) samples and can consist of a single physical layer frame or multiple physical layer frames.

When you set the `NumInputStreams` property to a value greater than 1, the data fields generated from different input streams are merged using the round-robin technique.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `dvbs2xWaveformGenerator`

info	Characteristic information about object
flushFilter	Flush transmit filter

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object
isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Generate DVB-S2X Waveform for Single-Input Stream

Generate a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) time-domain waveform for a single-input transport stream (TS) with a single physical layer (PL) frame per stream.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream.

```
numFrames = 1;
```

Create a DVB-S2X System object with pilot-aided PL.

```
s2xWaveGen = dvbs2xWaveformGenerator;
s2xWaveGen.PLSDecimalCode = 133;      % QPSK 13/45
                                      % All odd PLSDecimalCode values are pilot aided
disp(s2xWaveGen)
```

```
  dvbs2xWaveformGenerator with properties:

          StreamFormat: "TS"
        HasTimeSlicing: false
       NumInputStreams: 1
        PLSDecimalCode: 133
                   DFL: 18448
     PLScramblingIndex: 0
         RolloffFactor: 0.3500
   FilterSpanInSymbols: 10
      SamplesPerSymbol: 4

  Show all properties
```

Create the bit vector of information bits, `data`, of concatenated TS user packets.

```
syncBits = [0 1 0 0 0 1 1 1]';    % Sync byte for TS packet is 47 Hex
pktLen = 1496;                    % UP length without sync bits is 1496
numPkts = s2xWaveGen.MinNumPackets*numFrames;
txRawPkts = randi([0 1],pktLen,numPkts);
txPkts = [repmat(syncBits,1,numPkts); txRawPkts];
data = txPkts(:);
```

Generate a DVB-S2X time-domain waveform using the information bits, `data`.

```
txWaveform = s2xWaveGen(data);
```

Generate DVB-S2X Waveform Consisting of VL-SNR Frame

Generate a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) time-domain waveform for a single-input generic stream (GS) with multiple physical layer (PL) frames per stream.

The DVB-S2X waveform generated in this example consists of a very low signal to noise ratio (VL-SNR) frame of set 2.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream.

```
numFrames = 2;
```

Create a DVB-S2X System object and specify its properties.

```
s2xWaveGen = dvbs2xWaveformGenerator;
s2xWaveGen.StreamFormat = "GS";
s2xWaveGen.PLSDecimalCode = 131;    % VL-SNR set 2
s2xWaveGen.CanonicalMODCODName = "BPSK 1/3";
s2xWaveGen.DFL = 5080;
s2xWaveGen.SamplesPerSymbol = 6;
disp(s2xWaveGen)
```

```
  dvbs2xWaveformGenerator with properties:

          StreamFormat: "GS"
        HasTimeSlicing: false
       NumInputStreams: 1
                   UPL: 0
         PLSDecimalCode: 131
    CanonicalMODCODName: "BPSK 1/3"
                   DFL: 5080
      PLScramblingIndex: 0
          RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 6
```

Create a bit vector of information bits for each stream.

```
data = randi([0 1],s2xWaveGen.DFL*numFrames,1,'int8');
```

Generate a DVB-S2X time-domain waveform using the information bits.

```
txWaveform = s2xWaveGen(data);
```

Get DVB-S2X Waveform Generator Information and Check Transmit Filter Delay

Get information from a `dvbs2xWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 2;
```

Create a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) System object and specify its properties. Use time slicing technique and variable coding and modulation configuration mode.

```
s2xWaveGen = dvbs2xWaveformGenerator();
s2xWaveGen.HasTimeSlicing = true;
s2xWaveGen.NumInputStreams = 2;
s2xWaveGen.PLSDecimalCode = [135 145];   % QPSK 9/20 and 8PSK 25/36
s2xWaveGen.DFL = [18048 44656];
s2xWaveGen.PLScramblingIndex = [0 1];
disp(s2xWaveGen)
```

```
  dvbs2xWaveformGenerator with properties:

          StreamFormat: "TS"
        HasTimeSlicing: true
       NumInputStreams: 2
         PLSDecimalCode: [135 145]
                    DFL: [18048 44656]
     PLScramblingIndex: [0 1]
          RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 4
                  ISSYI: false

  Show all properties
```

Get the characteristic information about the DVB-S2X waveform generator.

```
info(s2xWaveGen)
```

```
ans = struct with fields:
              FECFrame: {'normal'  'normal'}
      ModulationScheme: {'QPSK'  '8PSK'}
    LDPCCodeIdentifier: {'9/20'  '25/36'}
```

Create the bit vector of input information bits, `data`, of concatenated TS user packets for each input stream.

```
syncBits = [0 1 0 0 0 1 1 1]';       % Sync byte for TS packet is 47 Hex
pktLen = 1496;                       % UP length without sync bits is 1496
data =  cell(1, s2xWaveGen.NumInputStreams);
for i = 1:s2xWaveGen.NumInputStreams
    numPkts = s2xWaveGen.MinNumPackets(i)*numFrames;
    txRawPkts = randi([0 1], pktLen, numPkts);
    txPkts = [repmat(syncBits, 1, numPkts); txRawPkts];
    data{i} = txPkts(:);
end
```

Generate a DVB-S2X time-domain waveform using the information bits.

```
txWaveform = s2xWaveGen(data);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(s2xWaveGen)
```

```
ans = 40×1 complex

  -0.2412 - 0.0143i
  -0.2619 - 0.0861i
  -0.2726 - 0.1337i
  -0.2511 - 0.1597i
  -0.1851 - 0.1680i
  -0.0780 - 0.1602i
   0.0448 - 0.1288i
   0.1598 - 0.0751i
   0.2482 - 0.0049i
   0.3026 + 0.0702i
      ⋮
```

References

[1] ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

[2] ETSI Standard EN 302 307-2 V1.1.1(2015-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: DVB-S2 Extensions (DVB-S2X)*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- For all properties that support string and cell array input, code generation is only supported with cell array input.

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects
`dvbs2WaveformGenerator`

Functions
`dvbs2BitRecover`

Introduced in R2021a

etsiRicianChannel

Filter input signal through multipath ETSI frequency-flat Rician fading channel

Description

The `etsiRicianChannel` System object filters an input signal through a multipath European Telecommunication Standards Institute (ETSI) frequency-flat Rician fading channel. For more information on the `etsiRicianChannel` fading model, see “Channel Model Block Diagram” on page 4-32.

To filter an input signal using a multipath ETSI Rician fading channel:

1. Create the `etsiRicianChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?.

Creation

Syntax

```
chan = etsiRicianChannel
chan = etsiRicianChannel(Name,Value)
```

Description

`chan = etsiRicianChannel` creates a multipath ETSI frequency-flat Rician fading channel System object. This object filters a real or complex input signal through the multipath channel to obtain the channel-impaired signal.

`chan = etsiRicianChannel(Name,Value)` sets properties on page 4-26 using one or more name-value pairs. Enclose each property name in quotes. For example, `etsiRicianChannel("SampleRate",2)` sets the input signal sample rate to 2.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SampleRate` — Input signal sample rate

1 (default) | positive scalar

Input signal sample rate in Hz, specified as a positive scalar.

Data Types: `double`

`KFactor` — Rician *K*-factor
3 (default) | nonnegative nonzero scalar

Rician *K*-factor in dB, specified as a nonnegative nonzero scalar.

`KFactor` is the ratio of direct signal power to the total multipath power. For details, see “Channel Model Block Diagram” on page 4-32.

Data Types: `double`

`MaximumDopplerShift` — Maximum Doppler shift for channel path
`0.001` (default) | nonnegative scalar

Maximum Doppler shift for the channel path, specified as a nonnegative scalar. Units are in hertz.

When you set this property to `0`, the channel remains static for the entire input. You can use the `reset` object function to generate a new channel realization. The `MaximumDopplerShift` property value must be smaller than `SampleRate`/10.

Data Types: `double`

`NumSinusoids` — Number of sinusoids used
48 (default) | positive integer

Number of sinusoids used to model the fading process, specified as a positive integer.

Data Types: `double`

`RandomStream` — Source of random number stream
`"Global stream"` (default) | `"mt19937ar with seed"`

Source of random number stream, specified as one of these options.

- `"Global stream"` — The current global random number stream is used for normally distributed random number generation. In this case, the `reset` object function resets the channel filters only.
- `"mt19937ar with seed"` — The mt19937ar algorithm is used for normally distributed random number generation. In this case, the `reset` object function resets the channel filters and reinitializes the random number stream to the value of the `seed` property.

Data Types: `char` | `string`

`Seed` — Initial seed of mt19937ar random number stream
73 (default) | nonnegative integer

Initial seed of the mt19937ar random number stream generator algorithm, specified as a nonnegative integer.

Dependencies

To enable this property, set the `RandomStream` property to `"mt19937ar with seed"`.

Data Types: `double`

`Visualization` — Channel visualization
`"Off"` (default) | `"Impulse response"` | `"Frequency response"` | `"Impulse and frequency responses"` | `"Doppler spectrum"`

Channel visualization, specified as `"Off"`, `"Impulse response"`, `"Frequency response"`, `"Impulse and frequency responses"`, or `"Doppler spectrum"`.

Data Types: `char` | `string`

Usage

Syntax

```
y = chan(x)
[y,pathgains] = chan(x)
```

Description

`y = chan(x)` filters input signal `x` through a multipath ETSI frequency-flat Rician fading channel and returns the output signal in `y`.

`[y,pathgains] = chan(x)` returns the channel path gains of the underlying multipath ETSI frequency-flat Rician fading process in `pathgains`.

Input Arguments

x — Input signal

N_S-by-1 vector

Input signal, specified as an N_S-by-1 vector, where N_S is the number of samples.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

y — Output signal

N_S-by-1 vector

Output signal, returned as an N_S-by-1 vector of complex values with the same data precision as the input signal x on page 4-0 . N_S is the number of samples.

Data Types: `double`
Complex Number Support: Yes

pathgains — Path gains

N_S-by-1 vector

Path gains, returned as an N_S-by-1 vector of complex values with the same data precision as the input signal x on page 4-0 . N_S is the number of samples.

Data Types: `double`
Complex Number Support: Yes

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `etsiRicianChannel`

info Characteristic information about object

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object
isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Transmit Input Signal Through ETSI Rician Channel

Transmit an input signal through a European Telecommunication Standards Institute (ETSI) Rician channel model.

Define the channel configuration using an `etsiRicianChannel` System object and specify its properties.

```
chan = etsiRicianChannel;
chan.SampleRate = 2.9e6;
chan.KFactor = 4;
chan.MaximumDopplerShift = 30;
chan.NumSinusoids = 45;
disp(chan)
```

```
  etsiRicianChannel with properties:

            SampleRate: 2900000
               KFactor: 4
    MaximumDopplerShift: 30

  Use get to show all properties
```

Generate a QPSK-modulated input signal to pass through the channel.

```
txWaveform = pskmod(randi([0 3],chan.SampleRate,1),4);
```

Filter the signal through the Rician channel.

```
rxWaveform = chan(txWaveform);
```

Verify ETSI Rician Channel Outputs Using Two Random Number Generation Methods

Produce the same multipath European Telecommunication Standards Institute (ETSI) Rician fading channel response by using two different methods for random number generation. The multipath ETSI Rician fading channel System object includes two methods for random number generation. You can

use the current global stream or the mt19937ar algorithm with a specified seed. By interacting with the global stream, the System object can produce the same outputs from the two methods.

Create `etsiRicianChannel` System object, and then specify its properties. Set the random number generation method as the mt19937ar algorithm.

```
chan = etsiRicianChannel;
chan.SampleRate = 150000;
chan.KFactor = 2;
chan.MaximumDopplerShift = 10;
chan.RandomStream = "mt19937ar with seed";
chan.Seed = 80;
```

Modulate randomly generated data.

```
txWaveform = pskmod(randi([0 3],512,1),4);
```

Filter the modulated data by using the multipath Rician fading channel System object.

```
[ChanOut1,PathGains1] = chan(txWaveform);
```

Set the System object to use the global stream for random number generation.

```
release(chan);
chan.RandomStream = "Global stream";
```

Set the global stream to have the same seed that was specified when creating the multipath Rician fading channel System object.

```
rng(80)
```

Filter the modulated data by using the multipath Rician fading channel System object again.

```
[ChanOut2,PathGains2] = chan(txWaveform);
```

Verify that the channel and path gain outputs are the same for each of the two random number generation methods.

```
isequal(ChanOut1,ChanOut2)
```

```
ans = logical
   1
```

```
isequal(PathGains1,PathGains2)
```

```
ans = logical
   1
```

Plot Doppler Spectrum for ETSI Rician Fading Channel

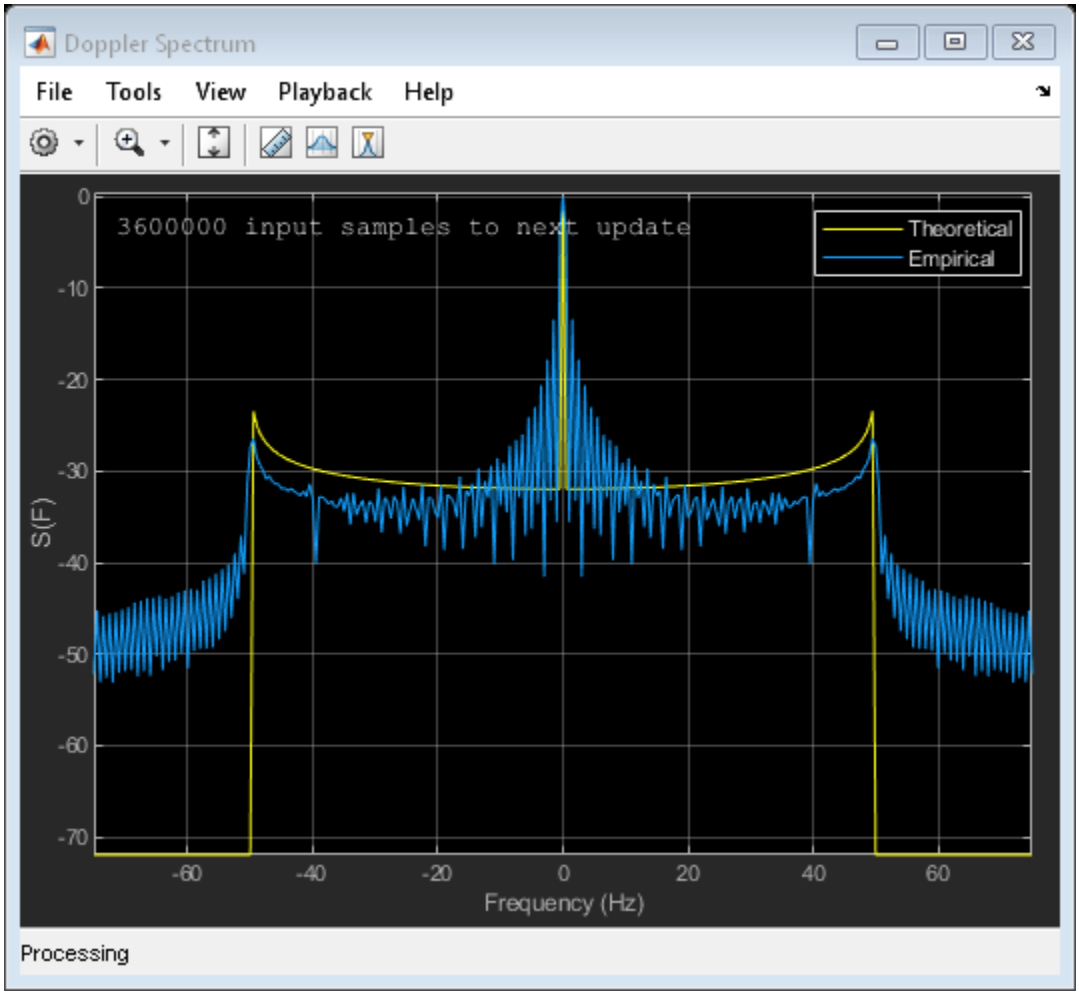
Create a multipath European Telecommunication Standards Institute (ETSI) Rician fading channel and display its Doppler spectrum.

Create `etsiRicianChannel` System object, and then specify its properties.

```
chan = etsiRicianChannel;
chan.SampleRate = 3.6e6;
chan.KFactor = 10;
chan.MaximumDopplerShift = 50;
chan.Visualization = "Doppler Spectrum";  % Jake's Doppler spectrum
```

Generate random binary data for `n` consecutive frames and pass the data through the multipath Rician fading channel.

```
n = 50;
for i = 1:n
  x = randi([0 1],3.6e6,1);
  y = chan(x);  % Spectrum visualization is updated only when the buffer is filled
                % Required samples to fill the buffer is mentioned in the scope
end
```



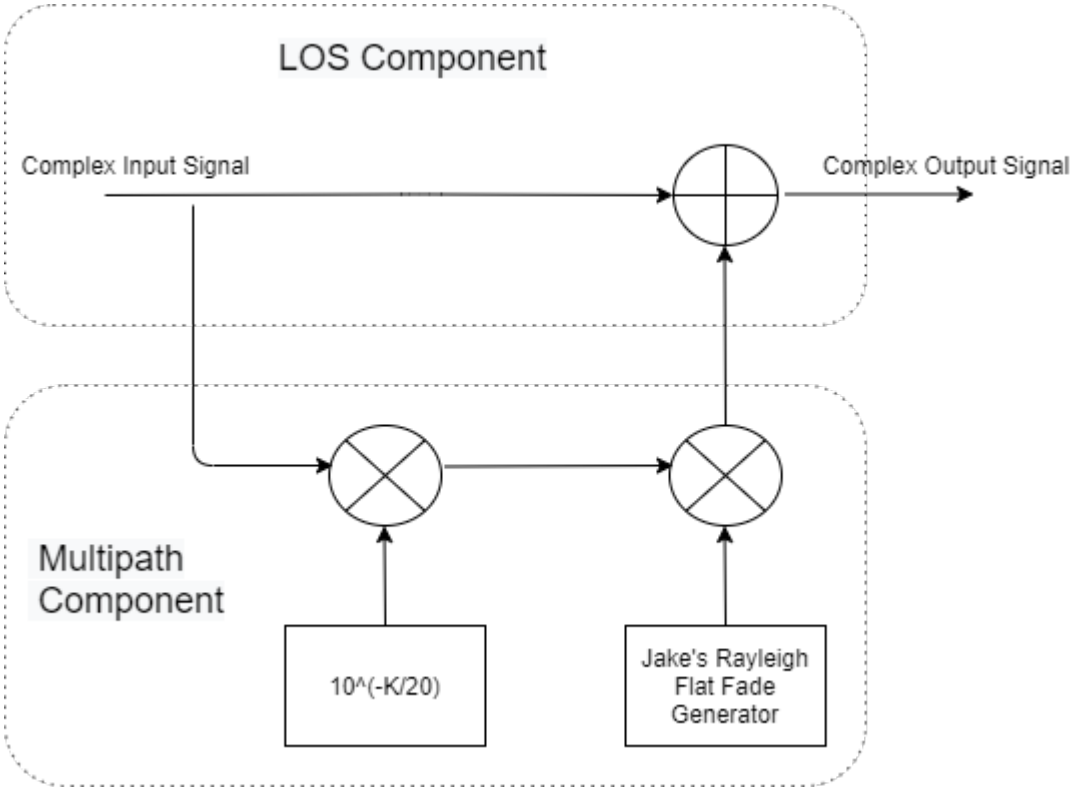
More About

Channel Model Block Diagram

The channel model block diagram provides an overview of the `etsiRicianChannel` System object, as specified in ETSI TS 101 376-5-5 V1.3.1 (2005-02) [1].

- The complex input signal is multiplied by a fixed gain and then by a complex Rayleigh fading gain. These actions form the multipath portion of the signal path. *K* is the Rician fade factor in dB.
- The multipath portion is then added to the direct signal component to form the Rician fading signal. This action forms the line-of-sight (LOS) component of the signal path.

  The coherent summation of many multipath components yield a classical Doppler spectrum for Rayleigh fading process, which when added to the direct path signal, forms the Rician fading signal.
- Noise samples can be subsequently added to the sum of the LOS component and multipath components.



Note The power of the complex output faded signal is $(1+1/K_f)$, where K_f is the "KFactor" on page 4-0 .

References

[1] ETSI TS 101 376-5-5 V1.3.1 (2005-02). *GEO-Mobile Radio Interface Specifications (Release 1); Part 5: Radio interface physical layer specifications; Sub-part 5: Radio Transmission and Reception; GMR-1 05.005.*

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Code generation is available only when the `Visualization` property is `"Off"`.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects

`comm.AWGNChannel` | `comm.RayTracingChannel` | `comm.RayleighChannel` | `comm.RicianChannel`

Functions

`doppler`

Introduced in R2021a

ccsdsTMWaveformGenerator

Generate CCSDS TM waveform

Description

The `ccsdsTMWaveformGenerator` System object generates a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) time-domain waveform. The object implements the waveform generation aspects of CCSDS standard blue books:

- CCSDS 131.0-B-3 — TM synchronization and channel coding [1]
- CCSDS 401.0-B-30 — Radio frequency and modulation systems [2]
- CCSDS 131.2-B-1 — Flexible advanced coding and modulation scheme for high rate TM applications [3]

Note The object supports waveform generation specified by the CCSDS TM synchronization and channel coding standard [1] and CCSDS flexible advanced coding and modulation scheme for high rate TM standard [3]. To obtain the waveform for either of the desired standard, set the `WaveformSource` property.

To generate a CCSDS TM waveform:

1. Create the `ccsdsTMWaveformGenerator` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?.

Creation

Syntax

```
tmWaveGen = ccsdsTMWaveformGenerator
tmWaveGen = ccsdsTMWaveformGenerator(Name,Value)
```

Description

`tmWaveGen = ccsdsTMWaveformGenerator` creates a default CCSDS TM waveform generator System object.

`tmWaveGen = ccsdsTMWaveformGenerator(Name,Value)` sets “Properties” on page 4-35 using one or more name-value pairs. For example, `ccsdsTMWaveformGenerator("WaveformSource","flexible advanced coding and modulation","ACMFormat",20)` specifies the CSSDS TM waveform source as flexible advanced coding and modulation standard with ACM format as 20 for the generated waveform.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

General

`WaveformSource` — CCSDS TM waveform source

`"synchronization and channel coding"` (default) | `"flexible advanced coding and modulation"`

CCSDS TM waveform source, specified as one of these values.

- `"synchronization and channel coding"` — Use this option to set the waveform to CCSDS TM synchronization and channel coding, as specified in CCSDS 131.0-B-3 [1].
- `"flexible advanced coding and modulation"` — Use this option to set the waveform to CCSDS flexible advanced coding and modulation for high rate TM applications, as specified in CCSDS 131.2-B-1 [3].

Data Types: `char` | `string`

`ACMFormat` — ACM format

`1` (default) | integer in the range [1, 27]

Adaptive coding and modulation (ACM) format, specified as an integer in the range [1, 27], as specified in CCSDS 131.2-B-1 Section 5.2.4 Table 5-2 [3].

Tunable: Yes

Dependencies

To enable this property, set the `WaveformSource` property to `"flexible advanced coding and modulation"`.

Data Types: `double` | `uint8`

`NumBytesInTransferFrame` — Number of bytes in one transfer frame

`223` (default) | integer in the range [1, 2048]

Number of bytes in one transfer frame, specified as an integer in the range [1, 2048].

Dependencies

To enable this property, one of these conditions should be satisfied:

- Set `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to `"none"`, `"convolutional"`, or `"LDPC"` on stream of sync marked transfer frame (SMTF).
- Set `WaveformSource` property to `"flexible advanced coding and modulation"`. In this case, the minimum number of `NumBytesInTransferFrame` is 223.

For other values of `ChannelCoding`, this `NumBytesInTransferFrame` property is calculated internally based on other properties.

Data Types: `double` | `uint16`

`HasRandomizer` — Option for randomizing data

`1` or `true` (default) | `0` or `false`

Option for randomizing the data, specified as a numeric or `logical` value of `1` (`true`) or `0` (`false`). Set this value to `1` (`true`) to randomize the data present in the channel access data unit (CADU).

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"`.

When you set the `ChannelCoding` property to `"LDPC"` and `IsLDPCOnSMTF` property to `1` (`true`), this property is not applicable, and is set to `1` (`true`).

Data Types: `double` | `logical`

`HasASM` — Option for inserting ASM

`1` or `true` (default) | `0` or `false`

Option for inserting attached sync marker (ASM), specified as a numeric or `logical` value of `1` (`true`) or `0` (`false`). Set this value to `1` (`true`) to indicate the data in CADU is attached with ASM.

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"`.

When you set the `ChannelCoding` property to `"LDPC"` and `IsLDPCOnSMTF` property to `1` (`true`), this property is not applicable, and is set to `1` (`true`).

Data Types: `double` | `logical`

`PCMFormat` — PCM format

`"NRZ-L"` (default) | `"NRZ-M"`

Pulse code modulation (PCM) format to select the PCM coding in the CCSDS TM waveform, specified as one of these values.

- `"NRZ-L"` — NRZ-level
- `"NRZ-M"` — NRZ-mark

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to `"BPSK"`, `"QPSK"`, `"8PSK"`, `"OPSK"`, or `"PCM/PSK/PM"`.

Data Types: `char` | `string`

Channel Coding

`ChannelCoding` — Forward error correction coding scheme

`"RS"` (default) | `"none"` | `"convolutional"` | `"concatenated"` | `"turbo"` | `"LDPC"`

Forward error correction coding scheme, specified as one of these values.

- `"none"`
- `"RS"`
- `"convolutional"`
- `"concatenated"`
- `"turbo"`
- `"LDPC"`

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"`.

Data Types: `char` | `string`

`NumBitsInInformationBlock` — Number of bits in turbo or LDPC message

`7136` (default) | `1784` | `3568` | `8920` | `1024` | `4096` | `16384`

Number of bits in the turbo or lower density parity check (LDPC) message, specified as one of these values.

- `1784`, `3568`, `7136`, or `8920` — Use one of these values when you set the `ChannelCoding` property to `"turbo"`.
- `1024`, `4096`, `16384`, or `7136` — Use one of these values when you set the `ChannelCoding` property to `"LDPC"`.

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to either `"turbo"` or `"LDPC"`.

Data Types: `double` | `uint8`

`ConvolutionalCodeRate` — Code rate of convolutional code

`"1/2"` (default) | `"2/3"` | `"3/4"` | `"5/6"` | `"7/8"`

Code rate of convolutional code, specified as a one of these values.

- `"1/2"`
- `"2/3"`
- `"3/4"`
- `"5/6"`
- `"7/8"`

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to either `"convolutional"` or `"concatenated"`.

When you set the `ChannelCoding` property to `"concatenated"`, the numeric value of the code rate also depends on the constituent Reed-Solomon (RS) code. You can obtain the actual numeric value for any code from the output field `ActualCodeRate` of the `info` object function.

Data Types: `char` | `string`

`CodeRate` — Code rate of turbo or LDPC code

`"1/2"` (for turbo code) (default) | `"7/8"` (for LDPC code) (default) | `"2/3"` | `"1/3"` | `"1/4"` | `"1/6"` | `"4/5"`

Code rate of turbo or LDPC code, specified as one of these values.

- `"1/2"`, `"1/3"`, `"1/4"`, or `"1/6"` — Use one of these values when you set the `ChannelCoding` property to `"turbo"`.
- `"1/2"`, `"2/3"`, `"4/5"`, or `"7/8"` — Use one of these values when you set the `ChannelCoding` property to `"LDPC"`.

Note When you set the `ChannelCoding` property to `"LDPC"` and the `NumBitsInInformationBlock` property to `7136`, the `CodeRate` must be `"7/8"`.

For an LDPC code, setting `CodeRate` to `7/8` implies an actual code rate numeric value of 223/255. You can obtain the actual numeric value for any code from the output field `ActualCodeRate` of the `info` object function.

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to either `"turbo"` or `"LDPC"`.

Data Types: `char` | `string`

`RSMessageLength` — Number of bytes in one RS message block

`223` (default) | `239`

Number of bytes in one RS message block, specified as `223` or `239`.

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to `"RS"` or `"concatenated"`.

Data Types: `double` | `uint8`

`RSInterleavingDepth` — Interleaving depth of RS code

`1` (default) | `2` | `3` | `4` | `5` | `8`

Interleaving depth of the RS code, specified as `1`, `2`, `3`, `4`, `5`, or `8`. The interleaving depth is the number of RS codewords in one code block.

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to `"RS"` or `"concatenated"`.

Data Types: `double` | `uint8`

`IsRSMessageShortened` — Option to shorten RS code

`0` or `false` (default) | `1` or `true`

Option to shorten the RS code, specified as a numeric or `logical` value of `0` (`false`) or `1` (`true`). Set this value to `1` (`true`) to shorten the RS code.

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to `"RS"` or `"concatenated"`.

Data Types: `double` | `logical`

`RSShortenedMessageLength` — Number of bytes in RS shortened message block

`223` (default) | integer in the range [1, `RSMessageLength`]

Number of bytes in the RS shortened message block, specified as an integer in the range [1, `RSMessageLength`].

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"`, the `ChannelCoding` property to `"RS"` or `"concatenated"`, and the `IsRSMessageShortened` property to `1` (`true`).

Data Types: `double` | `uint8`

`IsLDPCOnSMTF` — Option for using LDPC on stream of SMTF

`0` or `false` (default) | `1` or `true`

Option for using LDPC on the stream of a sync marked transfer frame (SMTF), specified as a numeric or `logical` value of `0` (`false`) or `1` (`true`). Set this value to `1` (`true`) to indicate LDPC on the stream of SMTF as specified in CCSDS 131.0-B-3 Section 8 of the TM synchronization and channel coding standard [1]. To indicate LDPC on the transfer frame, set this value to `0` (`false`).

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to `"LDPC"`.

Data Types: `double` | `logical`

`LDPCCodeBlockSize` — Number of LDPC codewords in LDPC code block of stream of SMTF

`1` (default) | integer in the range [1, 8]

Number of LDPC codewords in the LDPC code block of the stream of SMTF, specified as an integer in the range [1, 8].

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"`, the `ChannelCoding` property to `"LDPC"`, and the `IsLDPCOnSMTF` property to `true`.

Data Types: `double` | `uint8`

Digital Modulation and Filter

`Modulation` — Modulation scheme

`"QPSK"` (default) | `"BPSK"` | `"8PSK"` | `"OQPSK"` | `"GMSK"` | `"PCM/PSK/PM"` | `"PCM/PM/biphase-L"` | `"4D-8PSK-TCM"`

Modulation scheme used in CCSDS TC waveform, specified as one of these values.

- "QPSK"
- "BPSK"
- "8PSK"
- "OQPSK"
- "GMSK"
- "PCM/PSK/PM"
- "PCM/PM/biphase-L"
- "4D-8PSK-TCM"

Dependencies

To enable this property, set the WaveformSource property to "synchronization and channel coding".

Data Types: char | string

PulseShapingFilter — Pulse shaping filter

"root raised cosine" (default) | "none"

Pulse shaping filter, specified as "root raised cosine" or "none".

Dependencies

To enable this property, one of these conditions must be satisfied:

- Set WaveformSource property to "synchronization and channel coding" and the Modulation property to "BPSK", "QPSK", "8PSK", or "4D-8PSK-TCM".
- Set WaveformSource property to"flexible advanced coding and modulation".

Data Types: char | string

RolloffFactor — Roll-off factor of SRRC baseband filter

0.35 (default) | scalar in the range [0, 1]

Roll-off factor of the square root raised cosine (SRRC) baseband filter, specified as a scalar in the range [0, 1].

Note This property is not applicable when you set the PulseShapingFilter property to "none" for either value of the WaveformSource property.

Dependencies

To enable this property, one of these conditions must be satisfied:

- Set WaveformSource property to "synchronization and channel coding" and the Modulation property to either "BPSK", "QPSK", "8PSK", "OQPSK", or "4D-8PSK-TCM".
- Set WaveformSource property to "flexible advanced coding and modulation".

Data Types: double

FilterSpanInSymbols — Filter span in number of symbols

10 (default) | positive integer

Filter span in number of symbols, specified as a positive integer.

The `ccsdsTMWaveformGenerator` System object truncates the infinite impulse response of the ideal root raised cosine filter to this value.

Note This property is not applicable when you set the `PulseShapingFilter` property to `"none"` for either value of the `WaveformSource` property.

Dependencies

To enable this property, one of these conditions must be satisfied:

- Set `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to either `"BPSK"`, `"QPSK"`, `"8PSK"`, `"OQPSK"`, or `"4D-8PSK-TCM"`.
- Set `WaveformSource` property to `"flexible advanced coding and modulation"`.

Data Types: `double` | `uint32`

`BandwidthTimeProduct` — Bandwidth time product for GMSK modulator

`0.25` (default) | `0.5`

Bandwidth time product for the Gaussian minimum shift keying (GMSK) modulator, specified as `0.25` or `0.5`.

Dependencies

To enable this property, set `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to `"GMSK"`.

Data Types: `double`

`ModulationEfficiency` — Modulation efficiency of 4D-8PSK-TCM

`2` (default) | `2.25` | `2.5` | `2.75`

Modulation efficiency of 4D-8PSK trellis coded modulator (TCM), specified as `2`, `2.25`, `2.5`, or `2.75`. This property indicates the number of bits for each complex baseband symbol.

Dependencies

To enable this property, set `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to `"4D-8PSK-TCM"`.

Data Types: `double`

`SubcarrierWaveform` — Type of waveform to PSK-modulate NRZ data

`"sine"` (default) | `"square"`

Type of waveform to PSK-modulate the non-return-to-zero (NRZ) data, specified as `"sine"` or `"square"`.

Dependencies

To enable this property, set `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `char` | `string`

`ModulationIndex` — Modulation index in residual carrier phase modulation

`0.4` (default) | scalar in the range [0.2, 2]

Modulation index in the residual carrier phase modulation, specified as a scalar in the range [0.2, 2]. Units are in radians.

Dependencies

To enable this property, set `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to `"PCM/PSK/PM"` or `"PCM/PM/biphase-L"`.

Data Types: `double`

`SymbolRate` — Coded symbol rate

`2000` (default) | positive scalar

Coded symbol rate in Hz, specified as a positive scalar.

Dependencies

To enable this property, set `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `double`

`SubcarrierToSymbolRateRatio` — Ratio of subcarrier frequency to symbol rate

`4` (default) | integer in the range [1, 50]

Ratio of the subcarrier frequency to the symbol rate, specified as an integer in the range [1, 50].

Dependencies

To enable this property, set `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `double` | `uint8`

`SamplesPerSymbol` — Number of samples per symbol

`10` (default) | positive integer

Number of samples per symbol, specified as a positive integer.

This property is applicable for either input value of the `WaveformSource` property.

Dependencies

To enable this property, one of these conditions must be satisfied:

- Set the `Modulation` property to `"OQPSK"`, `"PCM/PSK/PM"`, or `"GMSK"`.
- Set the `PulseShapingFilter` to `"root raised cosine"`.

Data Types: `double` | `uint8`

`HasPilots` — Option for inserting pilot symbols

`0` or `false` (default) | `1` or `true`

Option for inserting pilot symbols within data, specified as a numeric or `logical` value of `0` (`false`) or `1` (`true`). Set this value to `1` (`true`) to indicate pilots are inserted, as described in CCSDS flexible advanced coding and modulation scheme for high rate TM standard [3].

Dependencies

To enable this property, set the `WaveformSource` property to `"flexible advanced coding and modulation"`.

Data Types: `double` | `logical`

`ScramblingCodeNumber` — Scrambling code number
`0` (default) | integer in the range [0, (2^{18} – 2)]

Scrambling code number for flexible advanced coding and modulation for high rate TM applications standard [3], specified as an integer in the range [0, (2^{18} – 2)].

`ScramblingCodeNumber` is used to randomize the complex baseband symbols.

Dependencies

To enable this property, set the `WaveformSource` property to `"flexible advanced coding and modulation"`.

Data Types: `double` | `uint32`

Read-Only

`NumInputBits` — Minimum number of bits required to generate waveform
integer

This property is read-only.

Minimum number of input bits to generate a waveform, returned as an integer.

The number of input bits must be an integer multiple of `NumInputBits`.

Data Types: `double`

`MinNumTransferFrames` — Minimum number of transfer frames for nonempty output
integer

This property is read-only.

Minimum number of transfer frames for a nonempty System object output, specified as an integer.

When you set the `WaveformSource` property to `"flexible advanced coding and modulation"`, or to `"synchronization and channel coding"` with the `IsLDPCOnSMTF` property set to `1` (`true`), System object output is empty until it has sufficient input to process through channel coding and modulation.

Data Types: `double`

Usage

Syntax

```
txWaveform = tmWaveGen(bits)
[txWaveform,encodedBits] = tmWaveGen(bits)
```

Description

`txWaveform = tmWaveGen(bits)` generates a CCSDS TM time-domain waveform for the corresponding input bits.

`[txWaveform,encodedBits] = tmWaveGen(bits)` also returns the bits obtained after TM synchronization and channel coding sublayer operations.

Input Arguments

`bits` — Information bits
binary-valued column vector

Information bits, in the form of transfer frames, specified as a binary-valued column vector. The length of this vector must be an integer multiple of the number of bits in one transfer frame. The `NumInputBits` property indicates the number of bits in one transfer frame.

Data Types: `double` | `int8` | `logical`

Output Arguments

`txWaveform` — Generated CCSDS TM time-domain waveform
column vector

Generated CCSDS TM time-domain waveform, returned as a column vector. This output is generated in the form of complex in-phase quadrature (IQ) samples.

Data Types: `double`

`encodedBits` — Output bits obtained after TM synchronization and channel coding sublayer operations
binary-valued column vector

Output bits obtained after TM synchronization and channel coding sublayer operations, returned as a binary-valued column vector.

Data Types: `double` | `int8` | `logical`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `ccsdsTMWaveformGenerator`

info	Characteristic information about object
flushFilter	Flush transmit filter

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object

isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Generate CCSDS TM Waveform for Synchronization and Channel Coding Scheme

Generate a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) waveform for the synchronization and channel coding standard, for multiple transfer frames. Visualize the waveform by using a spectrum plot.

Create a CCSDS TM System object. Set the waveform type as `synchronization and channel coding` with GMSK-modulated concatenated codes.

```
tmWaveGen = ccsdsTMWaveformGenerator;
tmWaveGen.WaveformSource = "synchronization and channel coding";
tmWaveGen.ChannelCoding = "concatenated";
tmWaveGen.Modulation = "GMSK";
tmWaveGen.RSMessageLength = 239;
tmWaveGen.RSInterleavingDepth = 2;
tmWaveGen.BandwidthTimeProduct = 0.5;
disp(tmWaveGen)
```

```
  ccsdsTMWaveformGenerator with properties:

          WaveformSource: "synchronization and channel coding"
           HasRandomizer: true
                 HasASM: true

   Channel coding properties:
            ChannelCoding: "concatenated"
    ConvolutionalCodeRate: "1/2"
          RSMessageLength: 239
      RSInterleavingDepth: 2
     IsRSMessageShortened: false

   Digital modulation and filter properties:
               Modulation: "GMSK"
     BandwidthTimeProduct: 0.5000
         SamplesPerSymbol: 10

  Use get to show all properties
```

Specify the number of transfer frames.

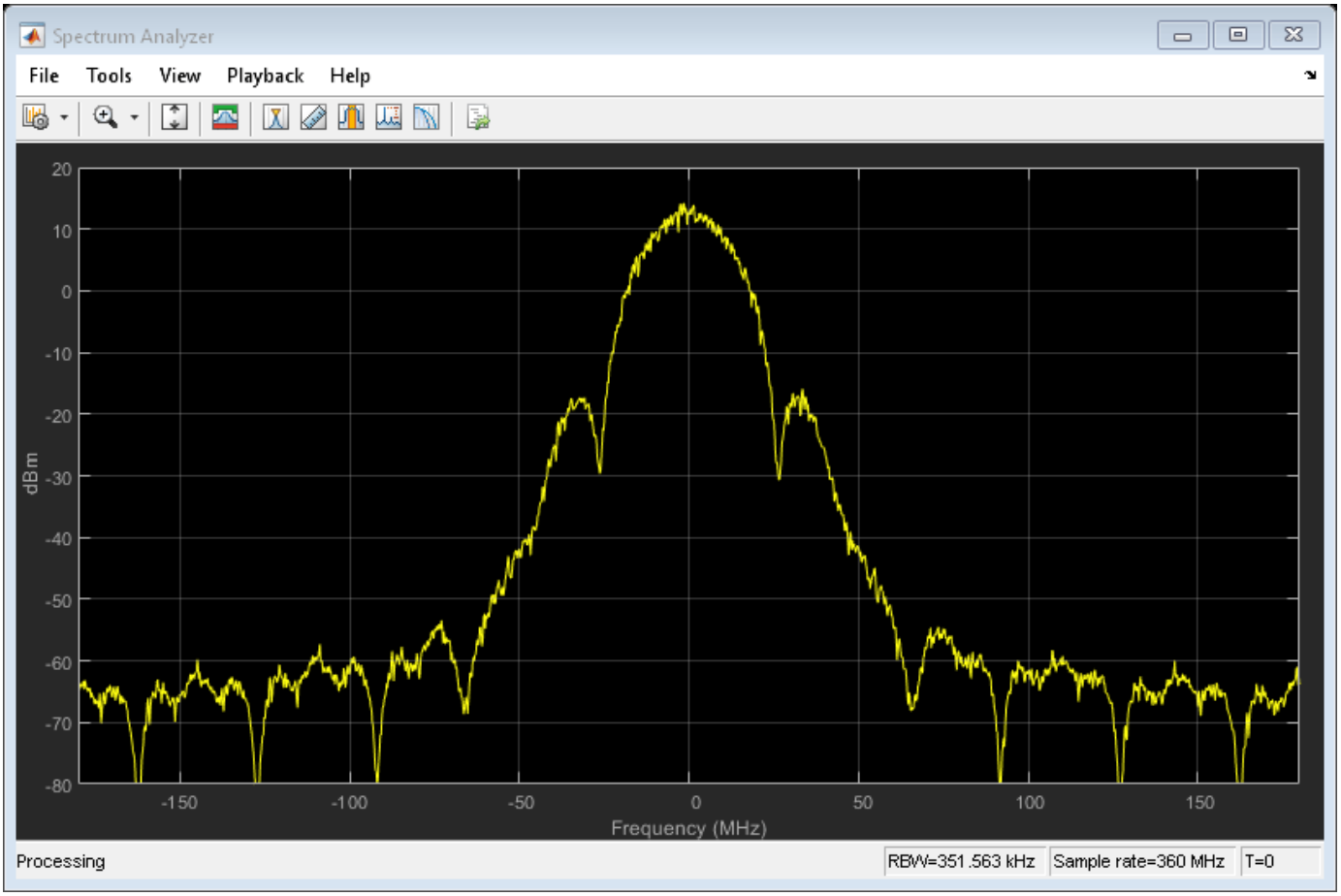
```
numTF = 15;
waveform = [];  % Initialize waveform as null
```

Generate the CCSDS TM waveform for the synchronization and channel coding standard by using multiple System object calls.

```
rng default    % For reproducible results
for iTF = 1:numTF
     bits = randi([0 1],tmWaveGen.NumInputBits,1);
     waveform = [waveform; tmWaveGen(bits)];
end
```

Create a `dsp.SpectrumAnalyzer` System object to display the frequency spectrum of the generated CCSDS TM time-domain waveform.

```
BW = 36e6;      % Typical satellite channel bandwidth
Fsamp = tmWaveGen.SamplesPerSymbol*BW;
scope = dsp.SpectrumAnalyzer('SampleRate',Fsamp,...
                             'AveragingMethod','Exponential');
scope(waveform)
```



Generate CCSDS TM Waveform for Flexible Advanced Coding and Modulation Scheme

Generate a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) waveform for the flexible advanced coding and modulation scheme for high rate TM applications standard, for one physical layer (PL) frame. Visualize the waveform by using a scatter plot.

Create a CCSDS TM System object, and then specify its properties.

```
tmWaveGen = ccsdsTMWaveformGenerator;
tmWaveGen.WaveformSource = "flexible advanced coding and modulation";
tmWaveGen.ACMFormat = 17;   % 16QPSK
tmWaveGen.PulseShapingFilter = "none";
disp(tmWaveGen)
```

```
  ccsdsTMWaveformGenerator with properties:

            WaveformSource: "flexible advanced coding and modulation"
                 ACMFormat: 17
    NumBytesInTransferFrame: 223

   Digital modulation and filter properties:
         PulseShapingFilter: "none"
                 HasPilots: false
       ScramblingCodeNumber: 0

  Use get to show all properties
```

Calculate the number of transfer frames in one PL frame.

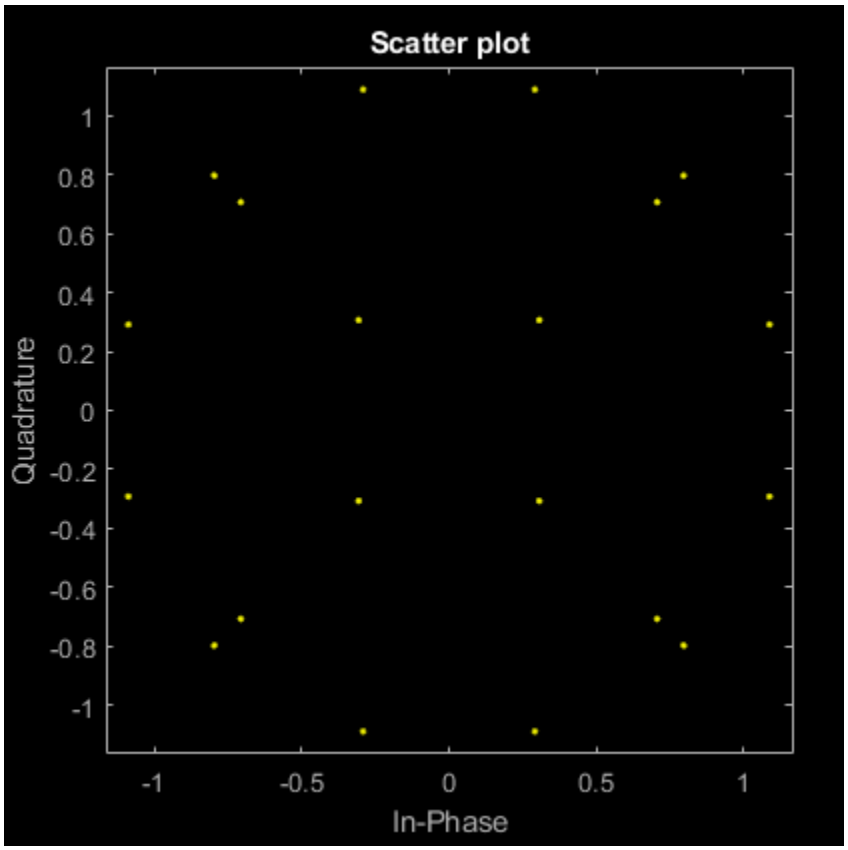
```
NumTFInOnePL = tmWaveGen.MinNumTransferFrames*16; % One PL frame consists of 16 codewords, as spe
waveform = [];  % Initialize waveform as null
```

Generate the CCSDS TM waveform for the flexible advanced coding and modulation scheme for high rate TM applications standard.

```
rng default    % For reproducible results
for iTF = 1:NumTFInOnePL
   bits = randi([0 1],tmWaveGen.NumInputBits,1);
   waveform = [waveform; tmWaveGen(bits)];
end
```

Display the scatter plot of the constellation for the generated waveform.

```
scatterplot(waveform);
legend off;
```



Get CCSDS TM Waveform Generator Information and Check Transmit Filter Delay

Get information from a `ccsdsTMWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

Create a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) System object. Set the waveform type as `synchronization and channel coding` with low-density parity-check (LDPC) channel coding. Display the properties.

```
tmWaveGen = ccsdsTMWaveformGenerator;
tmWaveGen.WaveformSource = "synchronization and channel coding";
tmWaveGen.ChannelCoding = "LDPC";
tmWaveGen.NumBitsInInformationBlock = 1024;
tmWaveGen.Modulation = "QPSK";
tmWaveGen.CodeRate = "1/2";
disp(tmWaveGen)
```

```
  ccsdsTMWaveformGenerator with properties:

              WaveformSource: "synchronization and channel coding"
               HasRandomizer: true
                      HasASM: true
                  PCMFormat: "NRZ-L"

   Channel coding properties:
```

```
                ChannelCoding: "LDPC"
    NumBitsInInformationBlock: 1024
                     CodeRate: "1/2"
                 IsLDPCOnSMTF: false

   Digital modulation and filter properties:
                   Modulation: "QPSK"
           PulseShapingFilter: "root raised cosine"
                RolloffFactor: 0.3500
          FilterSpanInSymbols: 10
             SamplesPerSymbol: 10

  Use get to show all properties
```

Specify the number of transfer frames.

```
numTF = 20;
```

Get the characteristic information about the CCSDS TM waveform generator.

```
info(tmWaveGen)
```

```
ans = struct with fields:
          ActualCodeRate: 0.5000
        NumBitsPerSymbol: 2
     SubcarrierFrequency: []
```

Generate the input bits for the CCSDS TM waveform generator, and then generate the waveform.

```
bits = randi([0 1], tmWaveGen.NumInputBits*numTF,1);
waveform = tmWaveGen(bits);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(tmWaveGen)
```

```
ans = 100×1 complex

  -0.0772 - 0.0867i
  -0.0751 - 0.0859i
  -0.0673 - 0.0788i
  -0.0549 - 0.0654i
  -0.0388 - 0.0469i
  -0.0200 - 0.0250i
   0.0002 - 0.0012i
   0.0208 + 0.0227i
   0.0405 + 0.0453i
   0.0587 + 0.0653i
      ⋮
```

References

[1] CCSDS 131.0-B-3. Blue Book. Issue 3. "TM Synchronization and Channel Coding." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, September 2017.

[2] CCSDS 401.0-B-30. Blue Book. Issue 30. "Radio Frequency and Modulation Systems - Part 1: Earth Stations and Spacecraft." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, February 2020.

[3] CCSDS 131.2-B-1. Blue Book. Issue 1. "Flexible Advanced Coding and Modulation Scheme for High Rate Telemetry Applications." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, March 2012.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`ccsdsTCIdealReceiver` | `ccsdsTCWaveform`

Objects

`ccsdsTCConfig`

Introduced in R2021a