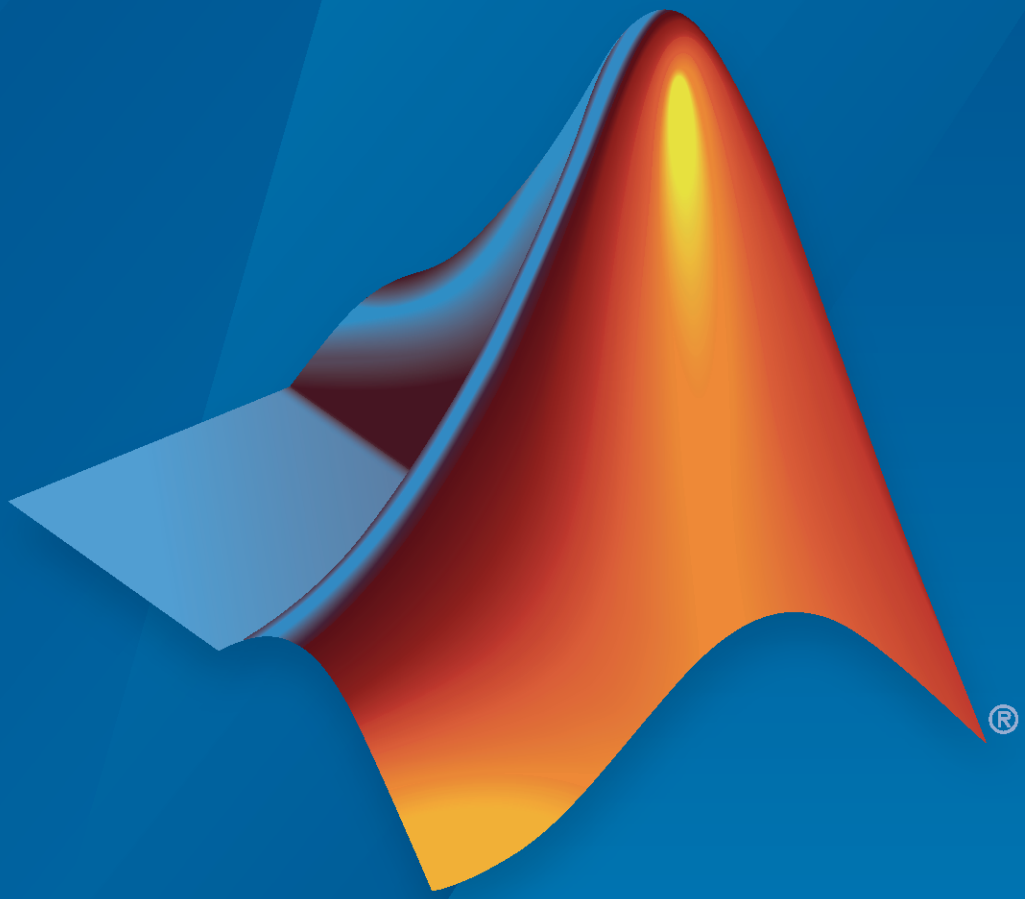
Bluetooth® Toolbox

Reference



MATLAB®

R2022b

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

Bluetooth® Toolbox Reference

Revision History

March 2022	Online only	New for Version 1.0 (Release 2022a)
September 2022	Online only	Revised for Version 1.1 (Release 2022b)

Contents

Functions

bleAngleEstimate

Estimate AoA or AoD of Bluetooth LE Signal

Syntax

```
angle = bleAngleEstimate(IQsamples,cfgAngle)
```

Description

`angle = bleAngleEstimate(IQsamples,cfgAngle)` estimates the angle of arrival (AoA) or angle of departure (AoD), `angle`, for the given in-phase and quadrature (IQ) samples, `IQsamples`, and Bluetooth® low energy (LE) angle estimation configuration object, `cfgAngle`.

Examples

Estimate AoA of the Bluetooth LE Signal

Create a Bluetooth LE angle estimation configuration object, specifying the values of antenna array size, slot duration, and antenna switching pattern.

```
cfgAngle = bleAngleEstimateConfig('ArraySize',2,'SlotDuration',2, ...
    'SwitchingPattern',[1 2])
```

```
cfgAngle =
  bleAngleEstimateConfig with properties:

            ArraySize: 2
       ElementSpacing: 0.5000
    EnableCustomArray: 0
         SlotDuration: 2
     SwitchingPattern: [1 2]
```

Set the IQ samples such that they define a connection data channel protocol data unit (PDU) with an AoA constant tone extension (CTE) of 2 μs slots, CTE time of 16 μs, and azimuth rotation of 70 degrees.

```
IQsamples = [0.8507+0.5257i;-0.5257 + 0.8507i;-0.8507 - 0.5257i; ...
    0.5257 - 0.8507i;0.8507+0.5257i;-0.5257 + 0.8507i; ...
    -0.8507 - 0.5257i;0.5257 - 0.8507i;-0.3561 + 0.9345i];
```

Estimate the AoA of the Bluetooth LE signal.

```
angle = bleAngleEstimate(IQsamples,cfgAngle)
```

```
angle = 70
```

Estimate AoA of Bluetooth LE Signal Received by Four-Element Uniform Circular Array

Create a Bluetooth LE angle estimation configuration object with custom array support enabled.

```
cfgAngle = bleAngleEstimateConfig(EnableCustomArray=1);
```

Specify the normalized element spacing, antenna switching pattern, and switch and sample slot duration of the angle estimation configuration.

```
cfgAngle.ElementPosition = [0 0 0 0; 0 0.5 1 0.5; 0 -0.5 0 0.5];
cfgAngle.SwitchingPattern = 1:4;
cfgAngle.SlotDuration = 2                                          % In microseconds
```

```
cfgAngle =
  bleAngleEstimateConfig with properties:

    EnableCustomArray: 1
      ElementPosition: [3x4 double]
         SlotDuration: 2
     SwitchingPattern: [1 2 3 4]
```

Set the IQ samples such that they define a connection data channel PDU with an AoA constant tone extension (CTE) of 2 µs slots, CTE time of 16 µs, and azimuth rotation of 70 degrees. Alternatively, you can use the IQ samples obtained by decoding the Bluetooth LE waveform.

```
IQsamples = [0.8507+0.5257i; -0.5257+0.8507i; -0.8507-0.5257i; ...
    0.5257-0.8507i; 0.8507+0.5257i; -0.5257+0.8507i; ...
    -0.8507-0.5257i; 0.5257-0.8507i; -0.3561+0.9345i; ...
    0.3561+0.9345i; 0.8507-0.5257i];
```

Estimate the AoA of the Bluetooth LE signal.

```
angle = bleAngleEstimate(IQsamples,cfgAngle)    % In degrees
```

```
angle = 2×1

  -54.2000
   37.2000
```

Input Arguments

`IQsamples` — IQ samples

complex-valued column vector

IQ samples, specified as a complex-valued column vector. This input corresponds to the 8 µs value of the reference period and slot duration.

Data Types: `single` | `double`

`cfgAngle` — Bluetooth LE angle estimation configuration object

`bleAngleEstimateConfig` object

Bluetooth LE angle estimation configuration object, specified as a `bleAngleEstimateConfig` object.

Output Arguments

`angle` — AoA or AoD

real number | two-element row vector of real numbers

AoA or AoD, returned as one of these values.

- Real number – This value is the estimated broadside angle. If *elevation* is 0, the estimated broadside angle represents the azimuth angle.
- Two-element row vector of real numbers in the form [*azimuth elevation*] – *azimuth* and *elevation* are the estimated azimuth angle and elevation angle in degrees, respectively.

The size of this output is equal to the size of the “ArraySize” on page 2-0 property of the `bleAngleEstimateConfig` object. If you set the EnableCustomArray property of the `bleAngleEstimateConfig` to true, the size of this output is equal to the two-element row vector.

Data Types: `double`

Version History

Introduced in R2020b

Includes support for custom antenna arrays

The function now supports custom antenna arrays for AoA and AoD estimation.

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed December 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.1. https://www.bluetooth.com/.

[3] Wooley, Martin. *Bluetooth Direction Finding: A Technical Overview*. Bluetooth Special Interest Group (SIG), Accessed December 6, 2021, https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`getElementPosition` | `getNumElements` | `bleWaveformGenerator` | `bleIdealReceiver`

Objects

`bleAngleEstimateConfig`

Topics

“Bluetooth Location and Direction Finding”

“Parameterize Bluetooth LE Direction Finding Features”
“Bluetooth LE Positioning by Using Direction Finding”
“Bluetooth LE Direction Finding for Tracking Node Position”

bleATTPDU

Generate Bluetooth LE ATT PDU

Syntax

```
attPDU = bleATTPDU(cfgATT)
```

Description

`attPDU = bleATTPDU(cfgATT)` generates a Bluetooth low energy (LE) attribute protocol data unit (ATT PDU) corresponding to the Bluetooth LE ATT PDU configuration object `cfgATT`.

Examples

Generate Bluetooth LE ATT PDUs

Generate two unique Bluetooth LE ATT PDUs of the type `'Read by type request'` and `'Error response'`.

Create a default Bluetooth LE ATT PDU configuration object.

```
cfgATT = bleATTPDUConfig;
```

Set the Bluetooth LE ATT PDU opcode to `'Read by type request'`.

```
cfgATT.Opcode = 'Read by type request'
```

```
cfgATT =
  bleATTPDUConfig with properties:

           Opcode: 'Read by type request'
      StartHandle: '0001'
        EndHandle: 'FFFF'
    AttributeType: '2800'
```

Generate a Bluetooth LE ATT PDU using the corresponding configuration object.

```
attPDU = bleATTPDU(cfgATT)
```

```
attPDU = 7x2 char array
    '08'
    '01'
    '00'
    'FF'
    'FF'
    '00'
    '28'
```

Create another Bluetooth LE ATT PDU configuration object, this time using the name-value pairs. Set the Bluetooth LE ATT PDU opcode to `'Error response'`.

```
cfgATT = bleATTPDUConfig('Opcode','Error response')

cfgATT =
  bleATTPDUConfig with properties:

             Opcode: 'Error response'
    RequestedOpcode: 'Read request'
    AttributeHandle: '0001'
       ErrorMessage: 'Invalid handle'
```

Generate a Bluetooth LE ATT PDU corresponding to this configuration object.

```
attPDU = bleATTPDU(cfgATT)

attPDU = 5x2 char array
    '01'
    '0A'
    '01'
    '00'
    '01'
```

Input Arguments

`cfgATT` — Bluetooth LE ATT PDU configuration object

`bleATTPDUConfig` object

Bluetooth LE ATT PDU configuration object, specified as a `bleATTPDUConfig` object.

Output Arguments

`attPDU` — Generated Bluetooth LE ATT PDU

character array

Generated Bluetooth LE ATT PDU, returned as a character array. Each row in this array is the hexadecimal representation of an octet.

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleATTPDUDecode`

Objects

`bleATTPDUConfig`

Topics

“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”

bleATTPDUDecode

Decode Bluetooth LE ATT PDU

Syntax

```
[status,cfgATT] = bleATTPDUDecode(attPDU)
```

Description

`[status,cfgATT] = bleATTPDUDecode(attPDU)` decodes the specified Bluetooth low energy (LE) attribute protocol data unit (ATT PDU), returning the corresponding Bluetooth LE ATT PDU configuration object, `cfgATT`, and the decoding status, `status`.

Examples

Decode Bluetooth LE ATT PDUs

Decode two unique Bluetooth LE ATT PDUs of type `'Read by type request'` and `'Error response'`.

Create a default Bluetooth LE ATT PDU configuration object.

```
cfgATT = bleATTPDUConfig;
```

Set the Bluetooth LE ATT PDU opcode to `'Read by type request'`.

```
cfgATT.Opcode = 'Read by type request'
```

```
cfgATT =
  bleATTPDUConfig with properties:

           Opcode: 'Read by type request'
      StartHandle: '0001'
        EndHandle: 'FFFF'
    AttributeType: '2800'
```

Generate a Bluetooth LE ATT PDU by using the corresponding configuration object.

```
attPDU = bleATTPDU(cfgATT)
```

```
attPDU = 7x2 char array
    '08'
    '01'
    '00'
    'FF'
    'FF'
    '00'
    '28'
```

Decode the generated Bluetooth LE ATT PDU. The returned status indicates decoding is successful.

```
[status,cfg] = bleATTPDUDecode(attPDU)

status =
  blePacketDecodeStatus enumeration

    Success


cfg =
  bleATTPDUConfig with properties:

           Opcode: 'Read by type request'
      StartHandle: '0001'
        EndHandle: 'FFFF'
    AttributeType: '2800'
```

Create another Bluetooth LE ATT PDU configuration object, this time using the name-value pairs. Set the Bluetooth LE ATT PDU opcode to `'Error response'`.

```
cfgATT = bleATTPDUConfig('Opcode','Error response')

cfgATT =
  bleATTPDUConfig with properties:

             Opcode: 'Error response'
    RequestedOpcode: 'Read request'
    AttributeHandle: '0001'
       ErrorMessage: 'Invalid handle'
```

Generate a Bluetooth LE ATT PDU by using the corresponding configuration object.

```
attPDU = bleATTPDU(cfgATT)

attPDU = 5x2 char array
    '01'
    '0A'
    '01'
    '00'
    '01'
```

Decode the generated Bluetooth LE ATT PDU. The returned status indicates decoding is successful.

```
[status,cfg] = bleATTPDUDecode(attPDU)

status =
  blePacketDecodeStatus enumeration

    Success


cfg =
  bleATTPDUConfig with properties:

             Opcode: 'Error response'
    RequestedOpcode: 'Read request'
    AttributeHandle: '0001'
```

```
       ErrorMessage: 'Invalid handle'
```

Decode Corrupted Bluetooth LE ATT PDU

Specify a Bluetooth LE ATT PDU containing corrupted data values.

```
attPDU = ['09'; '03'; '01'; '00'; '18'; '0D'];
```

Decode the specified Bluetooth LE ATT PDU. The returned status indicates that the decoding failed due to mismatched attribute data lengths. In case of failed decoding, the Bluetooth LE ATT PDU configuration object displays no properties.

```
[status,cfgATT] = bleATTPDUDecode(attPDU)
```

```
status =
  blePacketDecodeStatus enumeration

    MismatchAttributeDataLength


cfgATT =
  bleATTPDUConfig with properties:
```

Input Arguments

`attPDU` — Bluetooth LE ATT PDU

character vector | string scalar | numeric vector | character array

Bluetooth LE ATT PDU, specified as one of these values:

- Character vector — This vector represent octets in hexadecimal format.
- String scalar — This scalar represent octets in hexadecimal format.
- Numeric vector of elements in the range [0,255] — This vector represent octets in decimal format.
- *n*-by-2 character array — Each row represent an octet in hexadecimal format. *n* represents number of rows.

Data Types: `char` | `string` | `double`

Output Arguments

`cfgATT` — Bluetooth LE ATT PDU configuration object

`bleATTPDUConfig` object

Bluetooth LE ATT PDU configuration object, returned as a `bleATTPDUConfig` object.

`status` — Packet decoding status

nonpositive integer

Packet decoding status, returned as a nonpositive number of the type `blePacketDecodeStatus`. This value represents the result of an ATT PDU decoding. Each value of this output corresponds to a

member of the `blePacketDecodeStatus` enumeration class, which indicates the packet decoding status according to this table.

Value of `status`	Member of Enumeration Class	Decoding Status
0	`Success`	Packet decoding succeeded
–401	`UnsupportedATTOpcode`	Invalid ATT opcode
–402	`IncompleteATTPDU`	Incomplete ATT PDU
–403	`InvalidATTReqOpcodeInErrorResp`	Invalid requested opcode in "Error Response" PDUs
–404	`InvalidATTErrorCode`	Invalid error code
–405	`InvalidATTRxMTU`	Invalid received MTU
–406	`InvalidAttributeHandleRange`	Invalid attribute handle range
–407	`InvalidAttributeType`	Invalid attribute type flag
–408	`InvalidATTExecuteWriteFlag`	Invalid execute write flag
–409	`MismatchAttributeDataLength`	Length mismatches with actual length
–410	`InvalidATTDataFormat`	Invalid Data Format

An enumeration value other than `0` implies failed Bluetooth LE ATT PDU decoding. If the decoding fails, the `cfgATT` configuration object displays no output.

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleATTPDU`

Objects
`bleATTPDUConfig`

Topics
“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”

bleCTEIQSample

Perform IQ sampling on CTE field of Bluetooth LE packet

Syntax

```
iqSamples = bleCTEIQSample(cteSamples)
iqSamples = bleCTEIQSample(cteSamples,Name=Value)
```

Description

`iqSamples = bleCTEIQSample(cteSamples)` returns in-phase and quadrature (IQ) samples, `iqSamples`, by performing IQ sampling at each microsecond of the reference period and each sample slot of the constant tone extension (CTE), `cteSamples`.

`iqSamples = bleCTEIQSample(cteSamples,Name=Value)` specifies one or more optional name-value arguments. For example, `Mode="LE2M"` sets the Bluetooth low energy (LE) physical layer (PHY) reception mode to 2 Mbps.

Examples

Perform IQ Sampling on Bluetooth LE Waveform by Enabling CTE

Specify a connection-oriented protocol data unit (PDU) for the angle of departure (AoD) CTE. Set the slot duration and CTE time of the PDU to 2 microseconds and 16 microseconds, respectively.

```
pduHex = '1B820127';                % In hexadecimal
pdu = de2bi(hex2dec(pduHex),32)';  % In binary
```

Create a cyclic redundancy check (CRC) generator.

```
crcGen = comm.CRCGenerator('z^24+z^10+z^9+z^6+z^4+z^3+z+1', ...
    InitialConditions=de2bi(hex2dec('555551'),'left-msb',24), ...
    DirectMethod=true);
```

Append the CRC to the PDU.

```
pduCRC = crcGen(pdu);
```

Specify the samples per symbol and signal-to-noise ratio (SNR).

```
sps = 16;
snr = 30;   % In dB
```

Generate the Bluetooth LE transmit waveform.

```
txWaveform = bleWaveformGenerator(pduCRC,ChannelIndex=36,...
    DFPacketType='ConnectionCTE',SamplesPerSymbol=sps);
```

Pass the generated Bluetooth LE waveform through the additive white Gaussian noise (AWGN) channel.

```
rxWaveform = awgn(txWaveform,snr);
```

Segregate the noisy Bluetooth LE waveform corresponding to the CTE.

```
cteSamples = rxWaveform(end-16*sps+1:end);
```

Specify the slot duration and sample offset for IQ sampling. This example performs IQ sampling for a slot duration of 2 microseconds and takes the 8th sample after the occurrence of an antenna switch.

```
slotDuration = 2;   % In microseconds
sampleOffset = 8;
```

Perform IQ sampling on the Bluetooth LE waveform corresponding to the CTE.

```
iqSamples = bleCTEIQSample(cteSamples,SamplesPerSymbol=sps, ...
    SlotDuration=slotDuration,SampleOffset=sampleOffset)
```

```
iqSamples = 9×1 complex

  -0.7048 - 0.5840i
   0.5755 - 0.6943i
   0.7004 + 0.5720i
  -0.5714 + 0.7014i
  -0.7077 - 0.5714i
   0.5792 - 0.6993i
   0.7035 + 0.5750i
  -0.5810 + 0.7012i
  -0.5748 + 0.6908i
```

Input Arguments

`cteSamples` — Number of CTE samples

complex-valued column vector

Number of CTE samples received, specified as a complex-valued column vector representing a time-domain signal. The vector size must be *Ns*-by-1, where *Ns* is the number of CTE samples. This table shows how the `Mode` and `SamplesPerSymbol` (sps) name-value arguments impact the value of *Ns*.

`Mode` Value	***Ns* Value**	**Multiple of `SamplesPerSymbol`**
`LE1M`	[16×sps, 160×sps]	8×sps
`LE2M`	[32×sps, 320×sps]	16×sps

Data Types: `double`
Complex Number Support: Yes

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `Mode='LE2M'` sets the Bluetooth LE PHY reception mode to 2 Mbps.

`Mode` — PHY reception mode
`'LE1M'` (default) | `'LE2M'`

PHY reception mode, specified as `'LE1M'` or `'LE2M'`.

Data Types: `char` | `string`

`SamplesPerSymbol` — Samples per symbol
8 (default) | positive integer

Samples per symbol, specified as a positive integer.

Data Types: `double`

`SlotDuration` — Switch and sample slot duration
2 (default) | 1

Switch and sample slot duration, specified as 1 or 2. Units are in microseconds.

Data Types: `double`

`SampleOffset` — Sample offset
4 (default) | positive integer

Sample offset, specified as a positive integer in a range that is dependent on the `Mode` name-value argument.

`Mode` Value	`SampleOffset` Value Range
`'LE1M'`	[(sps/8), 7×(sps/8)]
`'LE2M'`	[(sps/4), 7×(sps/4)]

Data Types: `double`

Output Arguments

`iqSamples` — IQ samples
complex-valued column vector

IQ samples, returned as a complex-valued column vector. This output corresponds to the 8 microseconds value of the reference period and `SlotDuration`.

Data Types: `double`

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleWaveformGenerator` | `bleIdealReceiver` | `bleAngleEstimate`

Objects

`bleAngleEstimateConfig`

Topics

“Bluetooth Packet Structure”
“Bluetooth LE Positioning by Using Direction Finding”
“Bluetooth LE Direction Finding for Tracking Node Position”
“Parameterize Bluetooth LE Direction Finding Features”
“Estimate Bluetooth LE Node Position”
“Bluetooth Location and Direction Finding”

bleGAPDataBlock

Generate Bluetooth LE GAP data block

Syntax

```
dataBlock = bleGAPDataBlock(cfgGAP)
```

Description

`dataBlock = bleGAPDataBlock(cfgGAP)` generates a Bluetooth low energy (LE) generic access profile (GAP) data block of the type advertising data (AD) or scan response data (SRD) corresponding to the Bluetooth LE GAP data block configuration object, `cfgGAP`.

Examples

Generate Bluetooth LE GAP AD Blocks

Generate three unique Bluetooth LE GAP AD blocks: first one with AD types `'Flags'` and `'Tx power level'`, the second one with AD types `'Advertising interval'` and `'Local name'` and the third one with AD type `'Flags'`. All the three GAP AD blocks have simultaneous support for Bluetooth LE and basic rate/enhanced data rate (BR/EDR) at the host.

Create a configuration object for a Bluetooth LE GAP AD block and specify the AD types as `'Flags'` and `'Tx power level'`. Set the values of LED discoverability to `'Limited'` and Tx power level to 45.

```
cfgGAP = bleGAPDataBlockConfig;
cfgGAP.AdvertisingDataTypes = {'Flags';'Tx power level'};
cfgGAP.LEDiscoverability = 'Limited';
cfgGAP.TxPowerLevel = 45;
```

Generate the Bluetooth LE GAP AD block from the corresponding configuration object.

```
dataBlock1 = bleGAPDataBlock(cfgGAP)
```

```
dataBlock1 = 6x2 char array
    '02'
    '01'
    '05'
    '02'
    '0A'
    '2D'
```

Create a configuration object for a Bluetooth LE GAP AD block, this time with the advertising data types as `'Advertising interval'` and `'Local name'`. Specify the values of the advertising interval as 48, the local name as `'MathWorks'` and the local name shortening as `true`.

```
cfgGAP = bleGAPDataBlockConfig('AdvertisingDataTypes', ...
    {'Advertising interval', ...
```

```
    'Local name'});
cfgGAP.AdvertisingInterval = 48;
cfgGAP.LocalName = 'MathWorks';
cfgGAP.LocalNameShortening = true;
```

Generate the Bluetooth LE GAP AD block from the corresponding configuration object.

```
dataBlock2 = bleGAPDataBlock(cfgGAP)
```

```
dataBlock2 = 15x2 char array
    '03'
    '1A'
    '30'
    '00'
    '0A'
    '08'
    '4D'
    '61'
    '74'
    '68'
    '57'
    '6F'
    '72'
    '6B'
    '73'
```

Create a configuration object for a Bluetooth LE GAP AD block with type `'Flags'`. Set the values of LE discoverability to `'Limited'`, BR/EDR support to `true`, and simultaneous support for LE and BR/EDR to `'Host'`.

```
cfgGAP = bleGAPDataBlockConfig;
cfgGAP.LEDiscoverability = 'Limited';
cfgGAP.BREDR = true;
cfgGAP.LE = 'Host';
```

Generate the Bluetooth LE GAP AD block from the corresponding configuration object.

```
dataBlock3 = bleGAPDataBlock(cfgGAP)
```

```
dataBlock3 = 3x2 char array
    '02'
    '01'
    '11'
```

Input Arguments

`cfgGAP` — Bluetooth LE GAP data block configuration object

`bleGAPDataBlockConfig` (default) | object

Bluetooth LE GAP data block configuration object, specified as a `bleGAPDataBlockConfig` object.

Output Arguments

`dataBlock` — Generated Bluetooth LE GAP data block
character array

Generated Bluetooth LE GAP data block, returned as a character array. Each row in this array is the hexadecimal representation of an octet.

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "Supplement to the Bluetooth Core Specification." CSS Version 7 (2016).

Extended Capabilities

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

See Also

Functions
`bleGAPDataBlockDecode`

Objects
`bleGAPDataBlockConfig`

Topics
“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”

bleGAPDataBlockDecode

Decode Bluetooth LE GAP data block

Syntax

```
[status,cfgGAP] = bleGAPDataBlockDecode(dataBlock)
```

Description

`[status,cfgGAP] = bleGAPDataBlockDecode(dataBlock)` decodes a Bluetooth low energy (LE) generic access profile (GAP) data block, `dataBlock`, of the type advertising data (AD) or scan response data (SRD), returning the decoding status, `status`, and the Bluetooth LE GAP data block configuration object, `cfgGAP`.

Examples

Decode Bluetooth LE GAP AD Blocks

Decode two unique Bluetooth LE GAP AD blocks: one with AD types `'Flags'` and `'Tx power level'` and the other with AD types `'Advertising interval'` and `'Local name'`.

Create a configuration object for a Bluetooth LE GAP AD block. Specify the AD types as `'Flags'` and `'Tx power level'`. Set the values of LE discoverability to `'Limited'` and Tx power level to `45`.

```
cfgGAP = bleGAPDataBlockConfig;
cfgGAP.AdvertisingDataTypes = {'Flags';'Tx power level'};
cfgGAP.LEDiscoverability = 'Limited';
cfgGAP.TxPowerLevel = 45
```

```
cfgGAP =
  bleGAPDataBlockConfig with properties:

    AdvertisingDataTypes: {2x1 cell}
       LEDiscoverability: 'Limited'
                   BREDR: 0
            TxPowerLevel: 45
```

Generate a Bluetooth LE GAP AD block by using the corresponding configuration object.

```
dataBlock1 = bleGAPDataBlock(cfgGAP);
```

Decode the generated Bluetooth LE GAP AD block. The returned status indicates decoding was successful.

```
[status,cfgGAP] = bleGAPDataBlockDecode(dataBlock1)
```

```
status =
  blePacketDecodeStatus enumeration
```

```
    Success


cfgGAP =
  bleGAPDataBlockConfig with properties:

    AdvertisingDataTypes: {2x1 cell}
       LEDiscoverability: 'Limited'
                   BREDR: 0
           TxPowerLevel: 45
```

Create another Bluetooth LE GAP AD block configuration object, this time specifying AD types `'Advertising interval'` and `'Local name'`. Set the values of advertising interval as `48`, local name as `'MathWorks'`, and local name shortening as `true`.

```
cfgGAP = bleGAPDataBlockConfig('AdvertisingDataTypes', ...
    {'Advertising interval','Local name'});
cfgGAP.AdvertisingInterval = 48;
cfgGAP.LocalName = 'MathWorks';
cfgGAP.LocalNameShortening = true

cfgGAP =
  bleGAPDataBlockConfig with properties:

    AdvertisingDataTypes: {2x1 cell}
               LocalName: 'MathWorks'
     LocalNameShortening: 1
     AdvertisingInterval: 48
```

Generate the Bluetooth LE GAP AD block by using the corresponding configuration object.

```
dataBlock2 = bleGAPDataBlock(cfgGAP);
```

Decode the generated BLE GAP AD block. The returned status indicates decoding was successful. View the output of `'status'` and `'cfgGAP'`.

```
[status,cfgGAP] = bleGAPDataBlockDecode(dataBlock2)

status =
  blePacketDecodeStatus enumeration

    Success


cfgGAP =
  bleGAPDataBlockConfig with properties:

    AdvertisingDataTypes: {2x1 cell}
               LocalName: 'MathWorks'
     LocalNameShortening: 1
     AdvertisingInterval: 48
```

Decode Corrupted Bluetooth LE GAP AD Block

Specify a Bluetooth LE GAP AD block containing corrupted data values.

```
dataBlock = ['010106010202'];
```

Decode the specified Bluetooth LE GAP AD block. The returned status indicates that the decoding failed because of the corrupted input Bluetooth LE GAP AD block. When decoding fails, the Bluetooth LE GAP AD block configuration object, `'cfgGAP'`, displays no properties.

```
[status,cfgGAP] = bleGAPDataBlockDecode(dataBlock)
```

```
status =
  blePacketDecodeStatus enumeration

    MismatchGAPADLength


cfgGAP =
  bleGAPDataBlockConfig with properties:
```

Input Arguments

`dataBlock` — Bluetooth LE GAP data block

character vector | string scalar | numeric vector | character array

Bluetooth LE GAP data block, specified as one of these values:

- Character vector — This vector represents octets in hexadecimal format.
- String scalar — This scalar represents octets in hexadecimal format.
- Numeric vector of elements in the range [0, 255] — This vector represents octets in decimal format.
- *n*-by-2 character array — Each row represents an octet in hexadecimal format.

Data Types: `char` | `string` | `double`

Output Arguments

`status` — Bluetooth LE GAP data block decoding status

nonpositive integer

Bluetooth LE GAP data block decoding status, returned as a nonpositive integer of the type `blePacketDecodeStatus`. This value represents the result of a Bluetooth LE GAP data block decoding. Each value of `status` corresponds to a member of the `blePacketDecodeStatus` enumeration class, which indicates the packet decoding status according to this table.

Value of `status`	Member of Enumeration Class	Decoding Status
0	`Success`	Packet decoding succeeded
-201	`InvalidGAPADLength`	GAP AD length is not valid

–202	`MismatchGAPADLength`	Received AD length does not match with actual length
–203	`UnsupportedGAPADType`	Advertising data type is not valid or not supported
–204	`InvalidGAPAdvertisingInterval`	Advertising interval is not valid
–205	`InvalidGAPConnectionIntervalRange`	Invalid connection interval
–206	`InvalidGAPConnectionIntervalMinimum`	Invalid interval minimum
–207	`InvalidGAPConnectionIntervalMaximum`	Invalid interval maximum

An enumeration value other than `0` implies failed Bluetooth LE GAP data block decoding. If the decoding fails, the `cfgGAP` configuration object displays no output.

`cfgGAP` — Bluetooth LE GAP data block configuration object
`bleGAPDataBlockConfig` | object

Bluetooth LE GAP data block configuration object, returned as a `bleGAPDataBlockConfig` object.

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "Supplement to the Bluetooth Core Specification." CSS Version 7 (2016).

Extended Capabilities

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

See Also

Functions
`bleGAPDataBlock`

Objects
`bleGAPDataBlockConfig`

Topics
“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”

bleL2CAPFrame

Generate Bluetooth LE L2CAP frame

Syntax

```
L2CAPFrame = bleL2CAPFrame(cfgL2CAP)
L2CAPFrame = bleL2CAPFrame(cfgL2CAP,SDU)
```

Description

`L2CAPFrame = bleL2CAPFrame(cfgL2CAP)` generates a Bluetooth low energy (LE) logical link control and adaptation protocol (L2CAP) frame, `L2CAPFrame`, for a given Bluetooth LE L2CAP configuration object, `cfgL2CAP`. Use this syntax to generate the signaling frames.

`L2CAPFrame = bleL2CAPFrame(cfgL2CAP,SDU)` additionally generates a Bluetooth LE L2CAP frame for the upper-layer payload service data unit (SDU), `SDU`, by using the specified Bluetooth LE L2CAP configuration object. Use this syntax to generate the data frames.

Examples

Generate Bluetooth LE L2CAP Signaling Command Frame

Create a default Bluetooth LE L2CAP configuration object.

```
cfgL2CAP = bleL2CAPFrameConfig
```

```
cfgL2CAP =
  bleL2CAPFrameConfig with properties:

          ChannelIdentifier: '0005'
                CommandType: 'Credit based connection request'
           SignalIdentifier: '01'
    SourceChannelIdentifier: '0040'
                      LEPSM: '001F'
        MaxTransmissionUnit: 23
          MaxPDUPayloadSize: 23
                    Credits: 1
```

Set the value of credits to `10`.

```
cfgL2CAP.Credits = 10;
```

Generate a Bluetooth LE L2CAP signaling command frame from the specified configuration object.

```
l2capFrame = bleL2CAPFrame(cfgL2CAP)
```

```
l2capFrame = 18x2 char array
    '0E'
    '00'
    '05'
```

```
    '00'
    '14'
    '01'
    '0A'
    '00'
    '1F'
    '00'
    '40'
    '00'
    '17'
    '00'
    '17'
    '00'
    '0A'
    '00'
```

Generate Bluetooth LE L2CAP Data Frames

Generate two unique Bluetooth LE L2CAP data frames: one with SDU from the attribute protocol (ATT) layer as `'0A0100'` and the other with an upper-layer payload SDU,`'0A01E2D3'`.

Create a default Bluetooth LE L2CAP configuration object.

```
cfgL2CAP = bleL2CAPFrameConfig
```

```
cfgL2CAP =
  bleL2CAPFrameConfig with properties:

          ChannelIdentifier: '0005'
                CommandType: 'Credit based connection request'
           SignalIdentifier: '01'
    SourceChannelIdentifier: '0040'
                      LEPSM: '001F'
        MaxTransmissionUnit: 23
          MaxPDUPayloadSize: 23
                    Credits: 1
```

Set the value of ATT channel identifier to `'0004'`.

```
cfgL2CAP.ChannelIdentifier = '0004';
```

Generate a BLE L2CAP data frame from `'cfgL2CAP'`, specifying the upper-layer payload SDU from the ATT layer as `'0A0100'`.

```
l2capFrame = bleL2CAPFrame(cfgL2CAP,"0A0100")
```

```
l2capFrame = 7x2 char array
    '03'
    '00'
    '04'
    '00'
    '0A'
    '01'
```

```
    '00'
```

Create another default Bluetooth LE L2CAP configuration object. Set the value of dynamic channel identifier to `'007A'`.

```
cfgL2CAP = bleL2CAPFrameConfig;
cfgL2CAP.ChannelIdentifier = '007A';
```

Generate a Bluetooth LE L2CAP data frame from `'cfgL2CAP'`, specifying the upper-layer payload SDU as `'0A01E2D3'`.

```
l2capFrame = bleL2CAPFrame(cfgL2CAP,['0A'; '01'; 'E2'; 'D3'])
```

```
l2capFrame = 10x2 char array
    '06'
    '00'
    '7A'
    '00'
    '04'
    '00'
    '0A'
    '01'
    'E2'
    'D3'
```

Input Arguments

`cfgL2CAP` — Bluetooth LE L2CAP configuration object

`bleL2CAPFrameConfig` object

Bluetooth LE L2CAP configuration object, specified as a `bleL2CAPFrameConfig` object.

`SDU` — Upper-layer payload

character vector | string scalar | numeric vector | character array

Upper-layer payload, specified as one of these types:

- Character vector — This vector represents octets in hexadecimal format.
- String scalar — This scalar represents octets in hexadecimal format.
- Numeric vector of elements in the range [0, 255] — This vector represents octets in decimal format.
- *n*-by-2 character array — Each row represents an octet in the hexadecimal format.

Data Types: `char` | `double` | `string`

Output Arguments

`L2CAPFrame` — Generated Bluetooth LE L2CAP frame

character array

Generated Bluetooth LE L2CAP frame, returned as a character array. Each row of the array represents an octet in the hexadecimal format. This value represents the output Bluetooth LE L2CAP frame.

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleL2CAPFrameDecode`

Objects

`bleL2CAPFrameConfig`

Topics

“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”
“Bluetooth LE L2CAP Frame Generation and Decoding”

bleL2CAPFrameDecode

Decode Bluetooth LE L2CAP frame

Syntax

```
[status,cfgL2CAP,SDU] = bleL2CAPFrameDecode(L2CAPFrame)
```

Description

`[status,cfgL2CAP,SDU] = bleL2CAPFrameDecode(L2CAPFrame)` decodes the specified Bluetooth low energy (LE) logical link control and adaptation protocol (L2CAP) frame, `L2CAPFrame`. The function returns the decoding status, `status`, the corresponding Bluetooth LE L2CAP configuration object, `cfgL2CAP`, and the upper-layer payload service data unit (SDU), `SDU`.

Examples

Decode Bluetooth LE L2CAP Data Frame with SDU

Create a default Bluetooth LE L2CAP configuration object. Set the value of channel identifier to `'0004'`.

```
cfgL2CAP = bleL2CAPFrameConfig
```

```
cfgL2CAP =
  bleL2CAPFrameConfig with properties:

          ChannelIdentifier: '0005'
                CommandType: 'Credit based connection request'
           SignalIdentifier: '01'
    SourceChannelIdentifier: '0040'
                      LEPSM: '001F'
        MaxTransmissionUnit: 23
          MaxPDUPayloadSize: 23
                    Credits: 1
```

```
cfgL2CAP.ChannelIdentifier = '0004';
```

Generate a Bluetooth LE L2CAP data frame from `'cfgL2CAP'`, specifying the upper-layer payload SDU from attribute protocol (ATT) layer as `'0A0100'`.

```
L2CAPFrame = bleL2CAPFrame(cfgL2CAP,"0A0100");
```

Decode the generated Bluetooth LE L2CAP data frame. The returned status indicates decoding was successful.

```
[status,cfgL2CAP,SDU] = bleL2CAPFrameDecode(L2CAPFrame)
```

```
status =
  blePacketDecodeStatus enumeration
```

```
    Success


cfgL2CAP =
  bleL2CAPFrameConfig with properties:

    ChannelIdentifier: '0004'


SDU = 3x2 char array
    '0A'
    '01'
    '00'
```

Decode Corrupted Bluetooth LE L2CAP Frame

Specify a Bluetooth LE L2CAP frame containing corrupted data values.

```
l2capFrame = ['090005000107040060005000'];
```

Decode the specified Bluetooth LE L2CAP frame. The returned status indicates that the decoding failed due to the corrupted input L2CAP frame. If the decoding fails, the output displays the reason of failure and the object displays no properties.

```
[status,cfgL2CAP,sdu] = bleL2CAPFrameDecode(l2capFrame)
```

```
status =
  blePacketDecodeStatus enumeration

    MismatchL2CAPHeaderLength


cfgL2CAP =
  bleL2CAPFrameConfig with properties:


sdu =

  1x0 empty char array
```

Input Arguments

`L2CAPFrame` — Bluetooth LE L2CAP frame

character vector | string scalar | numeric vector | character array

Bluetooth LE L2CAP frame, specified as one of these values:

- Character vector — This vector represents octets in hexadecimal format.
- String scalar — This scalar represents octets in hexadecimal format.
- Numeric vector of elements in the range [0, 255] — This vector represents octets in decimal format.

- *n*-by-2 character array — Each row represents an octet in hexadecimal format.

Data Types: `char` | `double` | `string`

Output Arguments

`status` — Packet decoding status

nonpositive integer

Packet decoding status, returned as a nonpositive integer of type `blePacketDecodeStatus`. This value represents the result of decoding a Bluetooth LE L2CAP frame. Each value of `status` corresponds to a member of the `blePacketDecodeStatus` enumeration class, which indicates the packet decoding status according to this table.

Value of `status`	Member of Enumeration Class	Decoding Status
0	`Success`	Packet decoding succeeded
–301	`InvalidL2CAPConnectionIntervalRange`	Invalid connection intervals
–302	`InvalidL2CAPPeripheralLatency`	Invalid Peripheral latency
–303	`InvalidLECredits`	Invalid low energy (LE) credits
–304	`L2CAPSegmentationUnsupported`	Segmentation is not supported
–305	`MismatchL2CAPHeaderLength`	Length mismatches with actual length
–306	`IncompleteL2CAPDataFrame`	L2CAP data frame is not sufficient to decode
–307	`InvalidL2CAPChannelIdentifier`	Invalid L2CAP channel identifier
–308	`InvalidL2CAPCommand`	Invalid L2CAP command code
–309	`InvalidL2CAPCommandRejectReason`	Invalid command reject reason code
–310	`InvalidL2CAPParameterUpdateResult`	Invalid parameters update result
-311	`InvalidL2CAPConnectionResult`	Invalid connection result code
-312	`IllegalL2CAPSignalIdentifier`	Illegal signal identifier in L2CAP
-313	`InvalidL2CAPConnectionIntervalMinimum`	Invalid interval minimum
-314	`InvalidL2CAPConnectionIntervalMaximum`	Invalid interval maximum
-315	`InvalidL2CAPConnectionTimeout`	Invalid connection timeout

-316	`InvalidLEPSM`	Invalid LE protocol/service multiplexer
-317	`InvalidL2CAPChannelMTU`	Invalid maximum transmission unit
-318	`InvalidL2CAPChannelMPS`	Invalid maximum PDU payload size
-319	`InvalidL2CAPSDULength`	Invalid SDU length
-320	`MismatchL2CAPSignalFrameLength`	Length mismatches with actual length
-321	`IncompleteL2CAPSignalFrame`	L2CAP signal frame is not valid or not sufficient

An enumeration value other than `0` implies failed Bluetooth LE ATT PDU decoding. If the decoding fails, the `cfgATT` object displays no output.

`cfgL2CAP` — Bluetooth LE L2CAP frame configuration object
`bleL2CAPFrameConfig` object

Bluetooth LE L2CAP frame configuration object, returned as a `bleL2CAPFrameConfig` object.

`SDU` — Upper-layer payload
character array

Upper-layer payload, returned as a character array. Each row represents an octet in hexadecimal format.

Data Types: `char` | `string`

Note From R2022a, this function uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

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

See Also

Functions
`bleL2CAPFrame`

Objects
`bleL2CAPFrameConfig`

Topics
“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”
“Bluetooth LE L2CAP Frame Generation and Decoding”

bleLLAdvertisingChannelPDU

Generate Bluetooth LE LL advertising channel PDU

Syntax

```
advLLPDU = bleLLAdvertisingChannelPDU(cfgLLAdv)
```

Description

`advLLPDU = bleLLAdvertisingChannelPDU(cfgLLAdv)` generates a Bluetooth low energy (LE) link layer (LL) advertising channel protocol data unit (PDU), `advLLPDU`, corresponding to the Bluetooth LE LL advertising channel PDU configuration object, `cfgLLAdv`.

Examples

Generate Bluetooth LE LL Advertising Channel PDUs

Generate two unique Bluetooth LE LL advertising channel PDUs: first one of the type `'Advertising indication'` using advertising data `'020106'` and the other of the type `'Connection indication'` using a set of data channels to be used.

Create a Bluetooth LE LL advertising channel PDU configuration object, `'cfgLLAdv'`, with the opcode as `'Advertising indication'` by using advertising data `'020106'`.

```
cfgLLAdv = bleLLAdvertisingChannelPDUConfig;
cfgLLAdv.AdvertisingData = '020106'
```

```
cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                  PDUType: 'Advertising indication'
         ChannelSelection: 'Algorithm1'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '0123456789AB'
          AdvertisingData: [3x2 char]
```

Generate the Bluetooth LE LL advertising channel PDU by using the corresponding configuration object. Display the PDU length in octets.

```
advLLpdu = bleLLAdvertisingChannelPDU(cfgLLAdv);
numel(advLLpdu)/8
```

```
ans = 14
```

Display the first octet of the generated Bluetooth LE LL advertising channel PDU.

```
advLLpdu(1:8)
```

```
ans = 8×1
```

```
     0
     0
     0
     0
     0
     0
     1
     0
```

Create another Bluetooth LE LL advertising channel PDU configuration object, this time using the name-value pairs. Set the Bluetooth LE LL advertising channel PDU opcode to `'Connection indication'`.

```
cfgLLAdv = bleLLAdvertisingChannelPDUConfig('PDUType','Connection indication')
```

```
cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                 PDUType: 'Connection indication'
       ChannelSelection: 'Algorithm1'
   AdvertiserAddressType: 'Random'
       AdvertiserAddress: '0123456789AB'
    InitiatorAddressType: 'Random'
       InitiatorAddress: '0123456789CD'
          AccessAddress: '01234567'
      CRCInitialization: '012345'
             WindowSize: 1
           WindowOffset: 0
     ConnectionInterval: 6
      PeripheralLatency: 0
      ConnectionTimeout: 10
           UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ... ]
           HopIncrement: 5
     SleepClockAccuracy: '251 to 500 ppm'
```

Specify the value of connection interval as `8` and the set of data channels as `[0 4 12 16 18 24 25]`.

```
cfgLLAdv.ConnectionInterval = 8;                % In milliseconds
cfgLLAdv.UsedChannels = [0 4 12 16 18 24 25]
```

```
cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                 PDUType: 'Connection indication'
       ChannelSelection: 'Algorithm1'
   AdvertiserAddressType: 'Random'
       AdvertiserAddress: '0123456789AB'
    InitiatorAddressType: 'Random'
       InitiatorAddress: '0123456789CD'
          AccessAddress: '01234567'
      CRCInitialization: '012345'
             WindowSize: 1
           WindowOffset: 0
     ConnectionInterval: 8
      PeripheralLatency: 0
```

```
        ConnectionTimeout: 10
             UsedChannels: [0 4 12 16 18 24 25]
             HopIncrement: 5
       SleepClockAccuracy: '251 to 500 ppm'
```

Generate the Bluetooth LE LL advertising channel PDU from the corresponding configuration object. Display the PDU length in octets.

```
advLLpdu = bleLLAdvertisingChannelPDU(cfgLLAdv);
numel(advLLpdu)/8
```

```
ans = 39
```

Display the first octet of the generated Bluetooth LE LL advertising channel PDU.

```
advLLpdu(1:8)
```

```
ans = 8×1

     1
     0
     1
     0
     0
     0
     1
     1
```

Input Arguments

`cfgLLAdv` — Bluetooth LE LL advertising channel PDU configuration object

`bleLLAdvertisingChannelPDUConfig` object

Bluetooth LE LL advertising channel PDU configuration object, specified as a `bleLLAdvertisingChannelPDUConfig` object.

Output Arguments

`advLLPDU` — Generated Bluetooth LE LL advertising channel PDU

binary column vector

Generated Bluetooth LE LL advertising channel PDU, returned as a binary column vector.

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleLLAdvertisingChannelPDUDecode`

Objects

`bleLLAdvertisingChannelPDUConfig`

Topics

“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”
“Bluetooth LE Link Layer Packet Generation and Decoding”

bleLLAdvertisingChannelPDUDecode

Decode Bluetooth LE LL advertising channel PDU

Syntax

```
[status,cfgLLAdv] = bleLLAdvertisingChannelPDUDecode(advLLPDU)
[status,cfgLLAdv] = bleLLAdvertisingChannelPDUDecode(advLLPDU,inputFormat)
```

Description

`[status,cfgLLAdv] = bleLLAdvertisingChannelPDUDecode(advLLPDU)` decodes the specified Bluetooth low energy (LE) link layer (LL) advertising channel protocol data unit (PDU), `advLLPDU`, returning the decoding status, `status`, and the corresponding Bluetooth LE LL advertising channel PDU configuration object, `cfgLLAdv`.

`[status,cfgLLAdv] = bleLLAdvertisingChannelPDUDecode(advLLPDU,inputFormat)` additionally specifies the format of the Bluetooth LE LL advertising channel PDU.

Examples

Decode Bluetooth LE LL Advertising Channel PDU in Bits

Create a default Bluetooth LE LL advertising channel PDU configuration object.

```
cfgLLAdv = bleLLAdvertisingChannelPDUConfig
```

```
cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                  PDUType: 'Advertising indication'
         ChannelSelection: 'Algorithm1'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '0123456789AB'
          AdvertisingData: [3x2 char]
```

Generate the BLE LL advertising channel PDU from the corresponding configuration object.

```
advLLpdu = bleLLAdvertisingChannelPDU(cfgLLAdv);
```

Decode the generated Bluetooth LE LL advertising channel PDU. The returned status indicates decoding is successful.

```
[status,cfgLLAdv] = bleLLAdvertisingChannelPDUDecode(advLLpdu)
```

```
status =
  blePacketDecodeStatus enumeration

    Success
```

```
cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                  PDUType: 'Advertising indication'
         ChannelSelection: 'Algorithm1'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '0123456789AB'
          AdvertisingData: [3x2 char]
```

Decode Bluetooth LE LL Advertising Channel PDU Given in Octets

Specify a sample Bluetooth LE LL advertising channel PDU in octets.

```
advLLpdu = 'C409AB8967452301020106A8F1DF';
```

Decode the specified Bluetooth LE LL advertising channel PDU by specifying `'InputFormat'` to `'octets'`. The returned status indicates decoding is successful.

```
[status, cfgLLAdv] = bleLLAdvertisingChannelPDUDecode(advLLpdu, ...
    'InputFormat','octets')
```

```
status =
  blePacketDecodeStatus enumeration

    Success


cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                  PDUType: 'Scan response'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '0123456789AB'
         ScanResponseData: [3x2 char]
```

Decode Corrupted Bluetooth LE LL Advertising Channel PDU

Specify a Bluetooth LE LL advertising channel PDU containing corrupted data values.

```
pdu = 'D409AB89674523010201';
```

Decode the specified Bluetooth LE LL advertising channel PDU. The returned status indicates that the decoding failed due to the corrupted input Bluetooth LE LL advertising channel PDU. In case of failed decoding, the reason of failure is indicated and the object displays no properties.

```
[status,cfgLLAdv] = bleLLAdvertisingChannelPDUDecode(pdu, ...
    'InputFormat','octets')
```

```
status =
  blePacketDecodeStatus enumeration
```

```
    CRCFailed

cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:
```

Input Arguments

advLLPDU — Bluetooth LE LL advertising channel PDU

character vector | string scalar | numeric vector | character array | binary vector

Bluetooth LE LL advertising channel PDU, specified as one of these types:

- Character vector — This vector represents octets in hexadecimal format.
- String scalar — This scalar represents octets in hexadecimal format.
- Numeric vector of elements in the range [0,255] — This vector represents octets in decimal format.
- *n*-by-2 character array — Each row represents an octet in hexadecimal format. *n* represents number of rows.
- Binary vector — This vector represents the Bluetooth LE LL advertising channel PDU bits.

Data Types: `char` | `string` | `double`

inputFormat — Bluetooth LE LL advertising channel PDU format

`'bits'` (default) | `'octets'`

Bluetooth LE LL advertising channel PDU format, specified as `'bits'` or `'octets'`. When specified as `'bits'`, this input is a binary vector. When specified as `'octets'`, this input is a numeric vector representing octets in the decimal format or a character array or a string scalar representing octets in hexadecimal format.

Data Types: `char` | `string` | `double`

Output Arguments

status — Bluetooth LE LL advertising channel PDU decoding status

nonpositive integer

Bluetooth LE LL advertising channel PDU decoding status, returned as a nonpositive number of type `blePacketDecodeStatus`. This value represents the result of Bluetooth LE LL advertising channel PDU decoding. Each value of this output corresponds to a member of the `blePacketDecodeStatus` enumeration class, which indicates the packet decoding status shown in this table.

Value of `status`	Member of Enumeration Class	Decoding Status
0	`Success`	Packet decoding succeeded
-1	`CRCFailed`	Link Layer PDU is corrupted
–2	`LLPDULengthMismatch`	Length field does not match with actual PDU length

–3	`InvalidLLPeripheralLatency`	Invalid Peripheral latency
–4	`InvalidLLConnectionTimeout`	Invalid connection timeout
–5	`InvalidLLWindowSize`	Invalid window size
–6	`InvalidLLWindowOffset`	Invalid window offset
–7	`InvalidLLConnectionInterval`	Invalid connection interval
–8	`InvalidLLChannelMap`	Invalid channel map
–51	`IncompleteLLAdvertisingChannelPDU`	Insufficient octets in advertising channel PDU
–52	`InvalidLLHopIncrement`	Invalid hop increment value
–53	`InvalidLLAdvertisingDataLength`	Invalid advertising data
–54	`InvalidLLScanResponseDataLength`	Invalid scan response data
–55	`UnsupportedLLAdvertisingPDUType`	Unsupported advertising channel PDU

An enumeration value other than `0` implies failed Bluetooth LE LL advertising channel PDU decoding. If the decoding fails, the `cfgLLAdv` argument displays no output.

`cfgLLAdv` — Bluetooth LE LL advertising channel PDU configuration object
`bleLLAdvertisingChannelPDUConfig` object

Bluetooth LE LL advertising channel configuration object, returned as a `bleLLAdvertisingChannelPDUConfig` object.

Note From R2022a, this function uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

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

See Also

Functions
`bleLLAdvertisingChannelPDU`

Objects
`bleLLAdvertisingChannelPDUConfig`

Topics
“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”
“Bluetooth LE Link Layer Packet Generation and Decoding”

bleLLDataChannelPDU

Generate Bluetooth LE LL data channel PDU

Syntax

```
dataLLpdu = bleLLDataChannelPDU(cfgLLData)
dataLLpdu = bleLLDataChannelPDU(cfgLLData,LLPayload)
```

Description

`dataLLpdu = bleLLDataChannelPDU(cfgLLData)` generates a Bluetooth low energy (LE) link layer (LL) data channel protocol data unit (PDU), `dataLLpdu`, for a given Bluetooth LE LL data channel configuration object, `cfgLLData`. Use this syntax to generate a Bluetooth LE LL control PDU.

`dataLLpdu = bleLLDataChannelPDU(cfgLLData,LLPayload)` generates a Bluetooth LE LL data channel PDU, `dataLLpdu`, containing the upper-layer payload `LLPayload` for a given Bluetooth LE LL data channel configuration object, `cfgLLData`. Use this syntax to generate a Bluetooth LE LL data PDU.

Examples

Generate Bluetooth LE LL Control PDU of Type Connection Update Indication

Create a default Bluetooth LE LL control PDU configuration object.

```
cfgControl = bleLLControlPDUConfig
```

```
cfgControl =
  bleLLControlPDUConfig with properties:

                Opcode: 'Connection update indication'
            WindowSize: 1
          WindowOffset: 0
    ConnectionInterval: 6
     PeripheralLatency: 0
     ConnectionTimeout: 10
               Instant: 0
```

Create a Bluetooth LE LL data channel PDU configuration object for a control PDU of type `'Connection update indication'`.

```
cfgLLData = bleLLDataChannelPDUConfig('LLID','Control', ...
    'ControlConfig',cfgControl)
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                 LLID: 'Control'
                 NESN: 0
```

```
       SequenceNumber: 0
             MoreData: 0
    CRCInitialization: '012345'
        ControlConfig: [1x1 bleLLControlPDUConfig]
```

Generate a Bluetooth LE LL data channel PDU of type `'Connection update indication'` by using the configuration object `'cfgLLData'`. Display the PDU length in octets.

```
dataLLpdu = bleLLDataChannelPDU(cfgLLData);
numel(dataLLpdu)/8
```

```
ans = 17
```

Display the first octet of the generated BLE LL data channel PDU.

```
dataLLpdu(1:8)
```

```
ans = 8×1

     1
     1
     0
     0
     0
     0
     0
     0
```

Generate Bluetooth LE LL Data PDU Using Upper-Layer Payload

Create a default Bluetooth LE LL data channel PDU configuration object.

```
cfgLLData = bleLLDataChannelPDUConfig
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (continuation fragment/empty)'
                NESN: 0
       SequenceNumber: 0
             MoreData: 0
    CRCInitialization: '012345'
```

Generate a Bluetooth LE LL data PDU by using the corresponding configuration object, `'cgLLData'` and the upper-layer payload `'030004000A0100'`. Display the PDU length in octets.

```
dataLLpdu = bleLLDataChannelPDU(cfgLLData,'030004000A0100');
numel(dataLLpdu)/8
```

```
ans = 12
```

Display the first octet of the generated BLE LL data PDU.

```
dataLLpdu(1:8)
```

```
ans = 8×1

     1
     0
     0
     0
     0
     0
     0
     0
```

Input Arguments

`cfgLLData` — Bluetooth LE LL data channel configuration object

`bleLLDataChannelPDUConfig` object

Bluetooth LE LL data channel configuration object, specified as a `bleLLDataChannelPDUConfig` object.

`LLPayload` — Upper-layer payload

character vector | string scalar | numeric vector | character array

Upper-layer payload, specified as one of these types:

- Character vector — This vector represents octets in hexadecimal format.
- String scalar — This scalar represents octets in hexadecimal format.
- Numeric vector of elements in the range [0,255] — This vector represents octets in decimal format.
- *n*-by-2 character array — Each row represents an octet in hexadecimal format. *n* represents total number of rows.

Data Types: `char` | `string` | `double`

Output Arguments

`dataLLpdu` — Generated Bluetooth LE LL data channel PDU

binary column vector

Generated Bluetooth LE LL data channel PDU, returned as a binary column vector.

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- When the `CRCInitialization` property of the `bleLLDataChannelPDUConfig` object is set to constant, code generation is possible.

See Also

Functions

`bleLLDataChannelPDUDecode`

Objects

`bleLLDataChannelPDUConfig` | `bleLLControlPDUConfig`

Topics

“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”
“Bluetooth LE Link Layer Packet Generation and Decoding”

bleLLDataChannelPDUDecode

Decode Bluetooth LE LL data channel PDU

Syntax

```
[status,cfgLLData,LLPayload] = bleLLDataChannelPDUDecode(dataLLPDU,CRCinit)
[status,cfgLLData,LLPayload] = bleLLDataChannelPDUDecode(dataLLPDU,CRCinit,
inputFormat)
```

Description

`[status,cfgLLData,LLPayload] = bleLLDataChannelPDUDecode(dataLLPDU,CRCinit)` decodes a Bluetooth low energy (LE) link layer (LL) data channel protocol data unit (PDU), `dataLLPDU`, returning the decoding status, `status`, the Bluetooth LE LL data channel PDU configuration object, `cfgLLData`, and the upper-layer payload, `LLPayload`. The `CRCinit` denotes the initialization value of cyclic redundancy check (CRC).

`[status,cfgLLData,LLPayload] = bleLLDataChannelPDUDecode(dataLLPDU,CRCinit, inputFormat)` additionally sets the format of the Bluetooth LE LL data channel PDU.

Examples

Decode Bluetooth LE LL Data Channel PDU Given in Bits

Create a default Bluetooth LE LL data channel PDU configuration object.

```
cfgLLData = bleLLDataChannelPDUConfig
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (continuation fragment/empty)'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
   CRCInitialization: '012345'
```

Set the LLID value to `'start fragment/complete'`. Initialize the cyclic redundancy check (CRC) value to `'ED321C'`.

```
cfgLLData.LLID = 'Data (start fragment/complete)';
cfgLLData.CRCInitialization = 'ED321C';
crcInit = 'ED321C';
```

Generate a Bluetooth LE LL data channel PDU with upper-layer payload in hexadecimal octets.

```
dataLLPDU = bleLLDataChannelPDU(cfgLLData,'030004000A0100');
```

Decode the generated Bluetooth LE LL data channel PDU. The returned status indicates decoding is successful.

```
[status,cfgLLData,llPayload] = bleLLDataChannelPDUDecode(dataLLPDU,crcInit)

status =
  blePacketDecodeStatus enumeration

    Success


cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (start fragment/complete)'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
    CRCInitialization: '012345'


llPayload = 7x2 char array
    '03'
    '00'
    '04'
    '00'
    '0A'
    '01'
    '00'
```

Decode Bluetooth LE LL Data Channel PDU Given in Octets

Specify a sample Bluetooth LE LL data channel PDU in octets.

```
dataLLPDU = '030C000100000600000000A000000FCD2A6';
```

Initialize the CRC value.

```
crcInit = 'ED323C';
```

Decode the specified Bluetooth LE LL data channel PDU by specifying the `'inputFormat'` to `'octets'`. The specified PDU is a control PDU. The returned status indicates decoding is successful. You can see the decoded configuration of the specified PDU in the `'ControlConfig'` property of `'cfgLLData'`.

```
[status,cfgLLData,llPayload] = bleLLDataChannelPDUDecode(dataLLPDU,crcInit, ...
    'InputFormat','octets')

status =
  blePacketDecodeStatus enumeration

    Success


cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Control'
```

```
                NESN: 0
       SequenceNumber: 0
             MoreData: 0
    CRCInitialization: '012345'
        ControlConfig: [1x1 bleLLControlPDUConfig]

llPayload =

  1x0 empty char array
```

Decode Corrupted Bluetooth LE LL Data Channel PDU

Specify a Bluetooth LE LL data channel PDU containing corrupted data values. Initialize the CRC value.

```
dataLLPDU = '040C000100000600000000A00';
crcInit = 'CD3234';
```

Decode the specified Bluetooth LE LL data channel PDU. The returned status indicates that the decoding failed due to the corrupted Bluetooth LE LL data channel PDU. If the decoding fails, the reason is indicated and the object displays no properties.

```
[status,cfgLLData,llPayload] = bleLLDataChannelPDUDecode(dataLLPDU,crcInit, ...
    'InputFormat','octets')

status =
  blePacketDecodeStatus enumeration

    CRCFailed


cfgLLData =
  bleLLDataChannelPDUConfig with properties:


llPayload =

  1x0 empty char array
```

Input Arguments

dataLLPDU — Bluetooth LE LL data channel PDU

character vector | string scalar | numeric vector | character array | binary vector

Bluetooth LE LL data channel PDU, specified as one of these values:

- Character vector — This vector represents octets in hexadecimal format.
- String scalar — This scalar represents octets in hexadecimal format.
- Numeric vector of elements in the range [0,255] — This vector represents octets in decimal format.
- *n*-by-2 character array — Each row represents an octet in hexadecimal format. *n* represents total number of rows.

- Binary vector — This vector represents Bluetooth LE LL data channel PDU bits.

Data Types: `char` | `string` | `double`

`CRCinit` — CRC initialization value
character vector | string scalar

CRC initialization value, specified as a 6-element character vector or a string scalar representing 3-octet hexadecimal value.

Data Types: `char` | `string`

`inputFormat` — Bluetooth LE LL data channel PDU format
`'bits'` (default) | `'octets'`

Bluetooth LE LL data channel PDU format, specified as `'bits'` or `'octets'`. When specified as `'bits'`, this input is a binary vector. When specified as `'octets'`, this input is a numeric vector representing octets in decimal format or a character array or a string scalar representing octets in hexadecimal format.

Data Types: `char` | `string` | `double`

Output Arguments

`status` — Bluetooth LE LL data channel PDU decoding status
nonpositive integer

Bluetooth LE LL data channel PDU decoding status, returned as a nonpositive number of type `blePacketDecodeStatus`. This value represents the result of a Bluetooth LE LL data channel PDU decoding. Each value of this output corresponds to a member of the `blePacketDecodeStatus` enumeration class, which indicates the packet decoding status as shown in this table.

Value of `status`	Member of Enumeration Class	Decoding Status
0	`Success`	Packet decoding succeeded
-1	`CRCFailed`	Link Layer PDU is corrupted
–2	`LLPDULengthMismatch`	Length field does not match with actual PDU length
–3	`InvalidLLPeripheralLatency`	Invalid Peripheral latency
–4	`InvalidLLConnectionTimeout`	Invalid connection timeout
–5	`InvalidLLWindowSize`	Invalid window size
–6	`InvalidLLWindowOffset`	Invalid window offset
–7	`InvalidLLConnectionInterval`	Invalid connection interval
–8	`InvalidLLChannelMap`	Invalid channel map
-101	`IncompleteLLDataChannelPDU`	Insufficient octets in data channel PDU

-102	`InvalidLLID`	Invalid LLID
-103	`UnsupportedLLOpCode`	Unsupported opcode
-104	`InvalidLLErrorCode`	Invalid error code
-105	`InvalidBluetoothVersion`	Invalid version
-106	`ExpectedNonZeroPayload`	Nonzero payload expected
-107	`MICNotSupported`	Payload contains MIC

An enumeration value other than `0` implies failed Bluetooth LE LL data channel PDU decoding. If decoding fails, the `cfgLLData` argument displays no output.

`cfgLLData` — Bluetooth LE LL data channel configuration object
`bleLLDataChannelPDUConfig` object

Bluetooth LE LL data channel configuration object, returned as a `bleLLDataChannelPDUConfig` object.

`LLPayload` — Upper-layer payload
array

Upper-layer payload, returned as a character array where each row is the hexadecimal representation of an octet.

Note From R2022a, this function uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- When the `CRCInitialization` property of the `bleLLDataChannelPDUConfig` object is set to constant, code generation is possible.

See Also

Functions
`bleLLDataChannelPDU`

Objects
`bleLLDataChannelPDUConfig` | `bleLLControlPDUConfig`

Topics
“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”
“Bluetooth LE Link Layer Packet Generation and Decoding”

bleIdealReceiver

Decode Bluetooth LE PHY waveform

Syntax

```
[bits,accessAddr] = bleIdealReceiver(waveform)
[bits,accessAddr] = bleIdealReceiver(waveform,Name,Value)
[___,IQsamples] = bleIdealReceiver(___)
```

Description

`[bits,accessAddr] = bleIdealReceiver(waveform)` decodes the Bluetooth low energy (LE) physical layer (PHY) waveform, `waveform`, generated by the `bleWaveformGenerator` function. The function returns the received bits, `bits`, and the access address information, `accesAddr`.

`[bits,accessAddr] = bleIdealReceiver(waveform,Name,Value)` also specifies options using one or more name-value pair arguments. For example, `'Mode','LE2M'` sets the PHY transmission mode of the desired Bluetooth LE waveform to `'LE2M'`.

`[___,IQsamples] = bleIdealReceiver(___)` additionally returns the in-phase and quadrature (IQ) samples, `IQsamples`, corresponding to constant tone extension (CTE).

Examples

Decode Bluetooth LE Waveform Using Default Settings

Create an input message column vector of length 1000 containing random binary-valued transmission bits.

```
txBits = randi([0 1],1000,1);
```

Generate a Bluetooth LE transmit waveform from the transmission bits by using the `bleWaveformGenerator` function.

```
txWaveform = bleWaveformGenerator(txBits);
```

Pass the transmit waveform through a noisy channel and obtain the received waveform.

```
snr = 30;                           % In dB
rxWaveform = awgn(txWaveform,snr);
```

Recover data bits from the received Bluetooth LE waveform. Check for the number of bit errors in the recovered bits. The returned value indicates that the function successfully decodes the Bluetooth LE waveform.

```
[rxBits,accessAddr] = bleIdealReceiver(rxWaveform);
numErr = biterr(txBits,rxBits)
```

```
numErr = 0
```

Decode Bluetooth LE Waveform for LE125K PHY Transmission Mode

Specify the values of PHY transmission mode, channel index and samples per symbol (sps).

```
phyMode = 'LE125K';
chanIndex = 2;
sps = 4;
```

Generate transmission bits containing random binary values.

```
txBits = randi([0 1],100,1);
```

Obtain the Bluetooth LE transmit waveform from the transmission bits and the specified name-value pairs using the `bleWaveformGenerator` function.

```
txWaveform = bleWaveformGenerator(txBits,'Mode',phyMode,...
    'SamplesPerSymbol',sps,'ChannelIndex',chanIndex);
```

Recover the data bits, and then compare them with the transmission bits. The recovered data bits match the transmission bits, indicating there are no errors in the decoded Bluetooth LE waveform.

```
rxBits = bleIdealReceiver(txWaveform,'Mode',phyMode,...
    'SamplesPerSymbol',sps,'ChannelIndex',chanIndex);
isequal(txBits,rxBits)
```

```
ans = logical
   1
```

Decode Bluetooth LE Waveform With CTE For Connectionless Scenario

Specify a connectionless advertising channel protocol data unit (PDU) for angle of arrival (AoA) CTE.

```
pduHex = '02049B0327';
pdu = de2bi(hex2dec(pduHex),40)';
```

Generate and append cyclic redundancy check (CRC) to the PDU.

```
crcGen = comm.CRCGenerator('z^24+z^10+z^9+z^6+z^4+z^3+z+1', ...
    'InitialConditions',de2bi(hex2dec('555551'),'left-msb',24), ...
    'DirectMethod',true);
pduCRC = crcGen(pdu);
```

Generate the Bluetooth LE transmit waveform.

```
txWaveform = bleWaveformGenerator(pduCRC,'ChannelIndex',36, ...
    'DFPacketType','ConnectionlessCTE');
```

Recover the data bits by demodulating, dewhitening, and IQ sampling for a slot duration of 2 μs.

```
[bits,accAddr,iqSamples] = bleIdealReceiver(txWaveform, ...
    'ChannelIndex',36,'DFPacketType','ConnectionlessCTE');
```

Input Arguments

`waveform` — Received time-domain signal

complex-valued vector

Received time-domain signal, specified as a complex-valued signal with size N_s-by-1, where N_s represents the number of received samples. The values of N_s depend on the `'Mode'` and `'SamplesPerSymbol'` input arguments, according to the constraints specified in this table.

Value of `'Mode'`	Value of N_s	Multiple of
`'LE1M'`	≥40×*sps*	*sps*
`'LE2M'`	≥48×*sps*	*sps*
`'LE500K'`	≥376×*sps*	2×*sps*
`'LE125K'`	≥376×*sps*	8×*sps*

Data Types: `double` | `single`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `bleIdealReceiver(waveform,'Mode','LE2M','ChannelIndex',36)`

`Mode` — PHY transmission mode

`'LE1M'` (default) | `'LE2M'` | `'LE500K'` | `'LE125K'`

PHY transmission mode, specified as the comma-separated pair consisting of `'Mode'` and `'LE1M'`, `'LE2M'`, `'LE500K'`, or `'LE125K'`. This value indicates the type of PHY that the function uses to decode the received Bluetooth LE waveform.

Data Types: `char` | `string`

`ChannelIndex` — Channel Index

37 (default) | integer in the range [0, 39]

Channel index, specified as the comma-separated pair consisting of `'ChannelIndex'` and an integer in the range [0, 39]. For data channels, specify this value in the range [0, 36]. This value is used by the data-dewhitening block.

Data Types: `double`

`SamplesPerSymbol` — Samples per symbol

8 (default) | positive integer

Samples per symbol, specified as the comma-separated pair consisting of `'SamplesPerSymbol'` and a positive integer. The object uses this value for Gaussian frequency shift keying (GFSK) modulation.

Data Types: `double`

`DFPacketType` — Type of direction finding packet

`'Disabled'` (default) | `'ConnectionlessCTE'` | `'ConnectionCTE'`

Type of direction finding packet, specified as the comma-separated pair consisting of `'DFPacketType'` and `'ConnectionlessCTE'`, `'ConnectionCTE'`, or `'Disabled'`.

Data Types: `char` | `string`

`SlotDuration` — Switch and sample slot duration
2 (default) | 1

Switch and sample slot duration, specified as the comma-separated pair consisting of `'SlotDuration'` and 1 or 2. Specify this value in microseconds.

Data Types: `double`

`WhitenStatus` — Data whiten status
`'On'` (default) | `'Off'`

Data whiten status, specified as `'On'` or `'Off'`. Set this value to `'On'` for the function to perform dewhitening on the demodulated bits (for `'LE1M'` and `'LE2M'` PHY) and the decoded bits (for `'LE500K'` and `'LE125K'` PHY).

Data Types: `char` | `string`

Output Arguments

`bits` — Payload bits
column vector

Payload bits, returned as a column vector of maximum length 260 bytes. This output represents the recovered information bits.

Data Types: `int8`

`accessAddr` — Access address information
32-bit column vector

Access address information, returned as a 32-bit column vector. The higher layers use this output to validate a packet.

Data Types: `int8`

`IQsamples` — IQ samples
complex-valued column vector

IQ samples, specified as a complex-valued column vector. This argument corresponds to the 8 µs value of the reference period and slot duration. If you set the value of “DFPacketType” on page 1-0 argument to `'ConnectionlessCTE'` or `'ConnectionCTE'`, then the function returns this output.

Data Types: `double`

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleWaveformGenerator` | `bluetoothWaveformGenerator` | `bluetoothIdealReceiver` | `bleAngleEstimate`

Objects

`bleAngleEstimateConfig` | `bluetoothWhiten`

Topics

“Bluetooth LE Waveform Reception Using SDR”

bluetoothPathLoss

Estimate path loss between Bluetooth BR/EDR or LE devices

Syntax

```
pathLoss = bluetoothPathLoss(distance,cfgPathLoss)
```

Description

`pathLoss = bluetoothPathLoss(distance,cfgPathLoss)` estimates the path loss between Bluetooth basic rate/enhanced data rate (BR/EDR) or low energy (LE) devices for the given path loss estimation configuration, `cfgPathLoss`. Input `distance` specifies the distance between the Bluetooth devices.

Examples

Estimate Path Loss Between Two Bluetooth Devices in Office Environment

Create a default Bluetooth path loss estimation configuration object. Set the signal propagation environment to `"Office"`.

```
cfgPathLoss = bluetoothPathLossConfig;
cfgPathLoss.Environment = "Office";
```

Specify the transmitter and receiver antenna gains in dBi.

```
cfgPathLoss.TransmitterAntennaGain = 5;
cfgPathLoss.ReceiverAntennaGain = 10;
```

Specify the distance between the two Bluetooth devices, in meters.

```
distance = 50;
```

Estimate the path loss in dB.

```
pathLoss = bluetoothPathLoss(distance,cfgPathLoss)
```

```
pathLoss = 146.5998
```

Estimate Path Loss Between a Bluetooth Transmitter and Two Receivers in Home Environment

Create a Bluetooth path loss estimation configuration object for a home environment.

```
cfgPathLoss = bluetoothPathLossConfig(Environment="Home");
```

Specify the distance between a Bluetooth transmitter and each of two Bluetooth receivers, in meters.

```
distance = [20 30];
```

Estimate the path loss in dB.

```
pathLoss = bluetoothPathLoss(distance,cfgPathLoss)
```

```
pathLoss = 1×2

   77.5839   94.8555
```

Input Arguments

`distance` — Distance between Bluetooth BR/EDR or LE devices

nonnegative scalar | row vector of nonnegative values

Distance between Bluetooth BR/EDR or LE devices, specified as a nonnegative scalar or a row vector of nonnegative values. Units are in meters.

Data Types: `double`

`cfgPathLoss` — Bluetooth BR/EDR or LE path loss configuration

`bluetoothPathLossConfig` object

Bluetooth BR/EDR or LE path loss configuration, specified as a `bluetoothPathLossConfig` object.

Output Arguments

`pathLoss` — Estimated path loss

scalar | row vector

Estimated path loss, returned as a scalar or a row vector. Units are in dB.

Data Types: `double`

Version History

Introduced in R2022b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed March 22, 2022. https://www.bluetooth.com/.

[2] Bluetooth Core Specification 5.3. Bluetooth Special Interest Group (SIG), Accessed March 22, 2022. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions
`bluetoothRange`

Objects
`bluetoothPathLossConfig`

blePositionEstimate

Estimate Bluetooth LE node position

Syntax

```
nodePosition = blePositionEstimate(locatorPosition,localizationMethod,
direction)
nodePosition = blePositionEstimate(locatorPosition,localizationMethod,
distance)
nodePosition = blePositionEstimate(locatorPosition,localizationMethod,
distance,direction)
```

Description

`nodePosition = blePositionEstimate(locatorPosition,localizationMethod, direction)` estimates the unknown Bluetooth low energy (LE) node position, `nodePosition`, for the known Bluetooth LE locator positions, `locatorPosition`, and the localization method, `localizationMethod`. When you specify the localization method as `'angulation'`, the function calculates `nodePosition` by using the angle of arrival (AoA) or angle of departure (AoD), `direction`, between each locator and node.

`nodePosition = blePositionEstimate(locatorPosition,localizationMethod, distance)` estimates the unknown Bluetooth LE node position by using the localization method as `'lateration'`. The `distance` input specifies the distance between each locator and Bluetooth LE node.

`nodePosition = blePositionEstimate(locatorPosition,localizationMethod, distance,direction)` estimates the unknown Bluetooth LE node position by using the localization method as `'direction-angle'`.

Examples

Estimate Bluetooth LE Transmitter Position in 2-D Network Using Angulation

Set the positions of the Bluetooth LE receivers (locators).

```
rxPosition = [-18 -40;-10 70];          % In meters
```

Specify the azimuth angle of the signal between each Bluetooth LE receiver and transmitter.

```
azimuthAngles = [29.0546 -60.2551];     % In degrees
```

Specify the localization method. Because the angle of the signal between each Bluetooth LE receiver and transmitter is known, set the localization method to `'angulation'`.

```
localizationMethod = "angulation";
```

Estimate the position of the Bluetooth LE transmitter. The actual position of the Bluetooth LE transmitter is at the origin: (0, 0).

```
txPosition = blePositionEstimate(rxPosition,localizationMethod, ...
    azimuthAngles)
```

```
txPosition = 2×1
10^-4 ×

    0.2374
    0.1150
```

Estimate Bluetooth LE Receiver Position in 3-D Network Using Lateration

Set the positions of the Bluetooth LE transmitters (locators).

```
txPosition = [-5 -15 -30 -12.5;8.6603 -15 -17.3205 -21.6506; ...
    -17.3205 21.2132 20 43.3013];                                % In meters
```

Specify the distance between each Bluetooth LE transmitter and receiver.

```
distance = [13.2964 33.4221 40 55.0728];                         % In meters
```

Specify the localization method. Because the distance between each Bluetooth LE transmitter and receiver is known, set the localization method to `'lateration'`.

```
localizationMethod = "lateration";
```

Estimate the position of the Bluetooth LE receiver. The actual position of the Bluetooth LE receiver is [-7.5, 4.33, -5].

```
rxPosition = blePositionEstimate(txPosition,localizationMethod, ...
    distance)
```

```
rxPosition = 3×1

   -7.5001
    4.3304
   -4.9999
```

Input Arguments

`locatorPosition` — Position of Bluetooth LE locators

column vector | matrix

Position of Bluetooth LE locators, specified as a two- or three- element column vector or a matrix of size 2-by-*N* or 3-by-*N*, where *N* is the number of Bluetooth LE locators in the network. Each column denotes the 2-D or 3-D position of the locator. Specify this input in meters.

Data Types: `double`

`localizationMethod` — Localization method

`"angulation"` | `"lateration"` | `"distance-angle"`

Localization method, specified as `"angulation"`, `"lateration"`, or `"distance-angle"`. This value specifies the localization method that the function uses to estimate the position of the Bluetooth LE node.

Data Types: `char` | `string`

`direction` — AoA or AoD between each Bluetooth LE locator and node
row vector | matrix

AoA or AoD between each Bluetooth LE locator and node, specified as a row vector of size 1-by-*N* or a matrix of size 2-by-*N*, where *N* is the number of Bluetooth LE locators in the network. The first row represents the azimuth angles between each locator and node, and the second row represents the elevation angles between each locator and node. Specify this value in degrees. The range of azimuth and elevation angles is [–180, 180] degrees and [–90, 90] degrees, respectively.

Data Types: `double`

`distance` — Distance between each Bluetooth LE locator and node
row vector

Distance between each Bluetooth LE locator and node, specified as a row vector of size 1-by-*N*, where *N* is the number of Bluetooth LE locators in the network. Specify this value in meters.

Data Types: `double`

Output Arguments

`nodePosition` — Estimated 2-D or 3-D position of Bluetooth LE node
column vector

Estimated 2-D or 3-D position of the Bluetooth LE node, returned as a two- or three- element column vector denoting the 2-D or 3-D position, respectively. Units are in meters.

Data Types: `double`

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

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

See Also

Functions
`bleAngleEstimate` | `bleWaveformGenerator` | `bleIdealReceiver`

Objects
`bleAngleEstimateConfig`

Topics
“Bluetooth Location and Direction Finding”
“Parameterize Bluetooth LE Direction Finding Features”
“Estimate Bluetooth LE Node Position”
“Bluetooth LE Positioning by Using Direction Finding”
“Bluetooth LE Direction Finding for Tracking Node Position”

bleWaveformGenerator

Generate Bluetooth LE PHY waveform

Syntax

```
waveform = bleWaveformGenerator(message)
waveform = bleWaveformGenerator(message,Name,Value)
```

Description

`waveform = bleWaveformGenerator(message)` generates `waveform`, a time-domain Bluetooth low energy (LE) physical layer (PHY) waveform by using the input information bits, `message`.

`waveform = bleWaveformGenerator(message,Name,Value)` also specifies options using one or more name-value pair arguments. For example, `'Mode','LE2M'` specifies the generating mode value of the desired Bluetooth LE waveform.

Examples

Generate and Visualize Bluetooth LE Waveform Using Default Settings

Create an input message column vector of length 2056 containing random binary values. Set the symbol rate to default value.

```
message = randi([0 1],2056,1);
symbolRate = 1e6;
```

Generate the Bluetooth LE waveform.

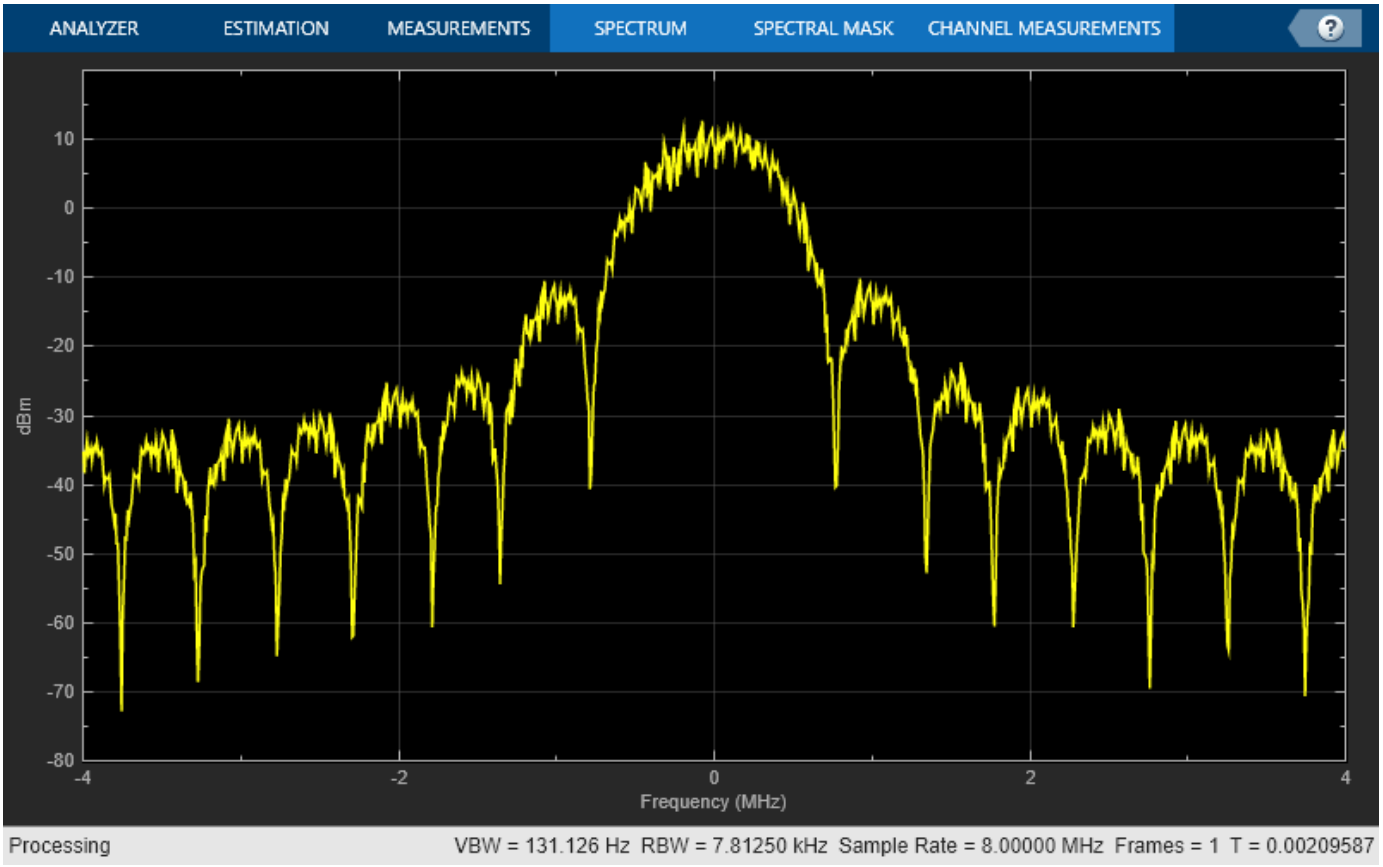
```
waveform = bleWaveformGenerator(message);
```

Create a spectrum analyzer System object to display the frequency spectrum of the generated Bluetooth LE waveform. Set the sample rate of the frequency spectrum.

```
scope = spectrumAnalyzer;
scope.SampleRate = 8*symbolRate;
```

Plot the Bluetooth LE waveform.

```
scope(waveform);
```



Generate and Visualize Bluetooth LE Waveform for LE125K PHY Mode

Create an input message column vector of length 640 containing random binary values.

```
message = randi([0 1],640,1);
```

Specify the values of generating mode, channel index, samples per symbol and access address. Set symbol rate to default value.

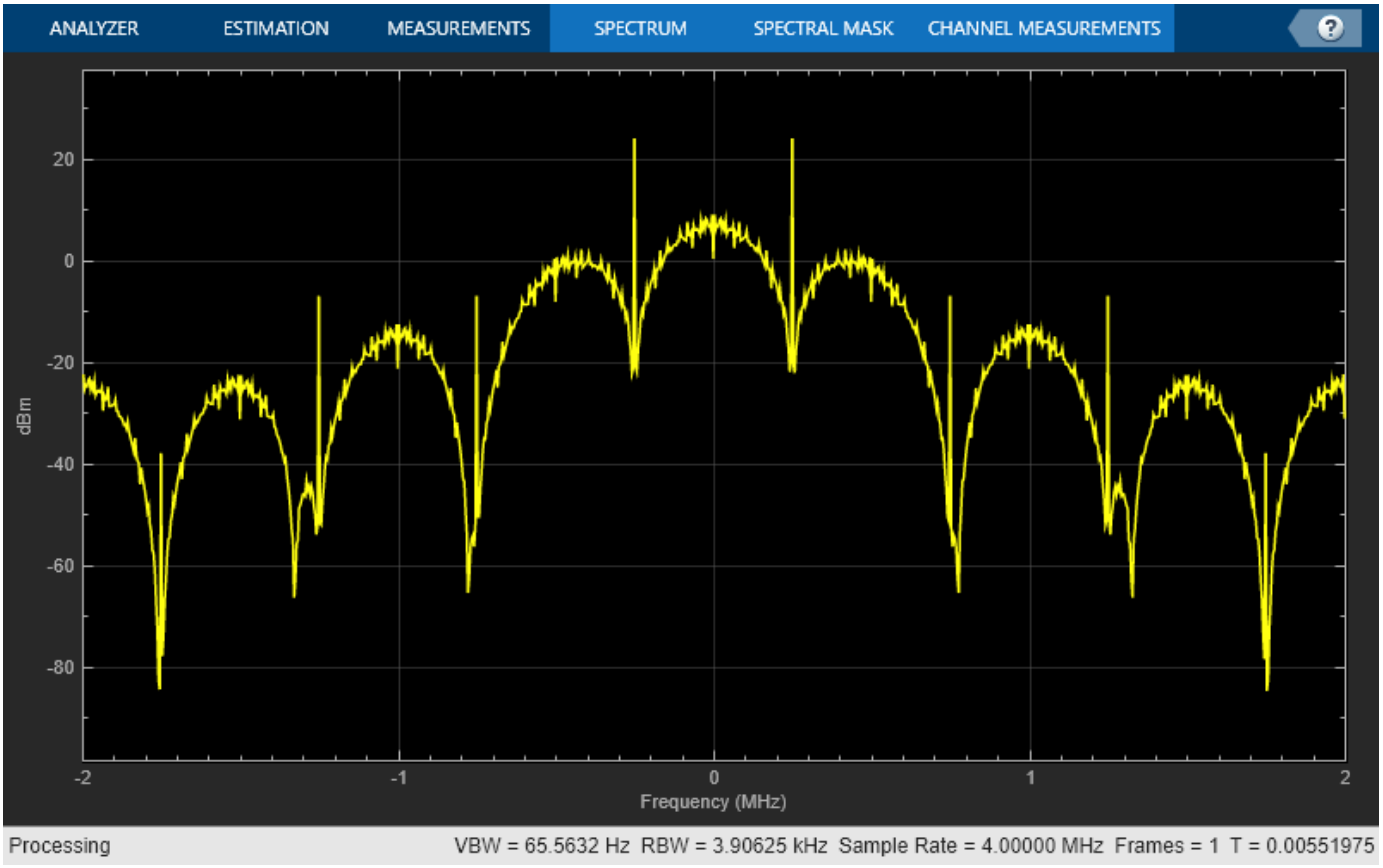
```
phyMode = "LE125K";
chanIdx = 2;
sps = 4;
accAdd = [1 1 1 1 0 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 1 1 1 0 1 ...
    0 1 0 1 1 0 0].';
symbolRate = 1e6;
```

Create a spectrum analyzer System object to display the frequency spectrum of the generated Bluetooth LE waveform. Set the sample rate of the frequency spectrum.

```
scope = spectrumAnalyzer;
scope.SampleRate = sps*symbolRate;
```

Generate and visualize the Bluetooth LE waveform.

```
waveform = bleWaveformGenerator(message,'Mode',phyMode, ...
    'SamplesPerSymbol',sps,'ChannelIndex',chanIdx,'AccessAddress',accAdd);
scope(waveform);
```



Generate Bluetooth LE Waveform With CTE For Connection-Oriented Scenario

Specify a connection data channel protocol data unit (PDU) for angle of departure (AoD) constant tone extension (CTE).

```
pduHex = '1B820127';                % Valid PDU in hexadecimal
pdu = de2bi(hex2dec(pduHex), 32)';
```

Generate and append cyclic redundancy check (CRC) to the PDU.

```
crcGen = comm.CRCGenerator('z^24+z^10+z^9+z^6+z^4+z^3+z+1', ...
    'InitialConditions',de2bi(hex2dec('555551'),'left-msb',24), ...
    'DirectMethod',true);
pduCRC = crcGen(pdu);
```

Generate the Bluetooth LE waveform.

```
waveform = bleWaveformGenerator(pduCRC,'ChannelIndex',36,...
    'DFPacketType','ConnectionCTE');
```

Input Arguments

`message` — Input message bits

binary-valued column vector

Input message bits, specified as a binary-valued column vector of numerical or logical values. This input contains the protocol data unit (PDU) and cyclic redundancy check (CRC) data. The maximum length of this value is 2088 bits.

Data Types: `double` | `logical`

Name-Value Pair Arguments

Note For information about connection CTE and connectionless CTE direction finding packet generation, see “Bluetooth Packet Structure”.

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `bleWaveformGenerator(message,'Mode','LE2M','ChannelIndex',36)`

`Mode` — PHY generating mode

`'LE1M'` (default) | `'LE2M'` | `'LE500K'` | `'LE125K'`

PHY generating mode, specified as the comma-separated pair consisting of `'Mode'` and `'LE1M'`, `'LE2M'`, `'LE500K'`, or `'LE125K'`.

Data Types: `char` | `string`

`ChannelIndex` — Channel Index

37 (default) | integer in the range [0, 39]

Channel index, specified as the comma-separated pair consisting of `'ChannelIndex'`and an integer in the range [0, 39]. For data channels, this value must be in the range [0, 36]. The data-whitening block uses this value to randomize the bits.

Data Types: `double`

`SamplesPerSymbol` — Samples per symbol

8 (default) | positive integer

Samples per symbol, specified as the comma-separated pair consisting of `'SamplesPerSymbol'` and a positive integer. The function uses this value for the Gaussian frequency-shift keying (GFSK) modulation.

Data Types: `double`

`AccessAddress` — Access address

`[0 1 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1]'` (default) | 32-bit column vector

Access address, specified as the comma-separated pair consisting of `'AccessAddress'` and a 32-bit column vector of numerical or logical values.

Data Types: `double` | `logical`

`DFPacketType` — Type of direction finding packet
`'Disabled'` (default) | `'ConnectionlessCTE'` | `'ConnectionCTE'`

Type of direction finding packet, specified as the comma-separated pair consisting of `'DFPacketType'` and `'ConnectionlessCTE'`, `'ConnectionCTE'`, or `'Disabled'`.

Data Types: `char` | `string`

`WhitenStatus` — Data whiten status
`'On'` (default) | `'Off'`

Data whiten status, specified as `'On'` or `'Off'`. To perform whitening on “message” on page 1-0 , set this value to `'On'`.

Data Types: `char` | `string`

Output Arguments

`waveform` — Output time-domain waveform
complex-valued column vector

Output time-domain waveform, returned as a complex-valued column vector of size *Ns*-by-1, where *Ns* represents the number of time-domain samples. The function returns this output in the form of complex in-phase quadrature (IQ) samples.

Data Types: `double`

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

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

See Also

Functions
`bleIdealReceiver` | `bluetoothIdealReceiver` | `bluetoothWaveformGenerator` | `bleAngleEstimate`

Objects
`bleAngleEstimateConfig` | `bluetoothWhiten`

Topics
“Bluetooth LE Waveform Generation and Visualization”
“Bluetooth LE Waveform Generation and Transmission Using SDR”
“Generate Bluetooth LE Waveform and Add RF Impairments”

bluetoothIdealReceiver

Decode Bluetooth BR/EDR PHY waveform

Syntax

```
[bits,decodedInfo] = bluetoothIdealReceiver(waveform,rxConfig)
[ ___ ,pktValidStatus,decodedCRC] = bluetoothIdealReceiver( ___ )
```

Description

`[bits,decodedInfo] = bluetoothIdealReceiver(waveform,rxConfig)` demodulates and decodes a synchronized time-domain Bluetooth basic rate/enhanced data rate (BR/EDR) waveform, `waveform`, generated by the `bluetoothWaveformGenerator` function for a given system configuration object, `rxConfig`. The function returns the decoded payload bits, `bits`, and decoded information, `decodedInfo`.

`[ ___ ,pktValidStatus,decodedCRC] = bluetoothIdealReceiver( ___ )` returns a flag, `pktValidStatus`, to indicate the validity of the received Bluetooth BR/EDR packet. The function also returns the decoded cyclic redundancy check (CRC), `decodedCRC`, of the received Bluetooth BR/EDR packet.

Examples

Demodulate and Decode Time-Domain Bluetooth BR/EDR Waveform

Demodulate and decode time-domain Bluetooth BR/EDR waveform by using the `bluetoothPhyConfig` object or `bluetoothWaveformConfig` object to extract the PHY information.

Use the `bluetoothPhyConfig` object to get PHY information

Create a default Bluetooth BR/EDR waveform configuration object.

```
txconfig = bluetoothWaveformConfig;
```

Create a random input bit vector to generate the payload. Generate the time-domain Bluetooth BR/EDR waveform by using the payload.

```
dataBits = randi([0 1],getPayloadLength(txconfig)*8,1);
waveform = bluetoothWaveformGenerator(dataBits,txconfig);
```

Create a default configuration object for the Bluetooth BR/EDR PHY with default settings. This object sends the PHY information to the Bluetooth ideal receiver.

```
rxConfig = bluetoothPhyConfig;
```

Demodulate and decode the Bluetooth BR/EDR waveform. The generated output displays the decoded bits and a structure containing the decoded information.

```
[bits,decodedInfo] = bluetoothIdealReceiver(waveform,rxConfig)
```

```
bits = 144×1

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1
      ⋮

decodedInfo = struct with fields:
                        LAP: [24x1 double]
                 PacketType: 'FHS'
    LogicalTransportAddress: [3x1 double]
          HeaderControlBits: [3x1 double]
              PayloadLength: 18
                       LLID: [2x1 double]
              FlowIndicator: 0
```

Use the `bluetoothWaveformConfig` object to get PHY information

Create a default Bluetooth BR/EDR waveform configuration object. Set the packet type to `'DM1'`and payload length to 10.

```
cfg = bluetoothWaveformConfig;
cfg.PacketType = 'DM1';
cfg.PayloadLength = 10;
```

Create a random input bit vector to generate the payload.

```
numBits = getPayloadLength(cfg)*8;
dataBits = randi([0 1],numBits,1);
```

Generate the time-domain Bluetooth BR/EDR waveform by using the payload.

```
waveform = bluetoothWaveformGenerator(dataBits,cfg);
```

Get the PHY information from the `bluetoothWaveformConfig` object function, `getPhyConfigProperties`.

```
rxConfig = getPhyConfigProperties(cfg);
```

Demodulate and decode the Bluetooth BR/EDR waveform. The generated output displays the decoded bits, a structure containing the decoded information, the packet status, and the decoded CRC.

```
[bits,decodedInfo,pktStatus,crc] = bluetoothIdealReceiver(waveform,rxConfig)
```

```
bits = 80×1

     0
     0
     0
     0
```

```
     0
     0
     0
     0
     1
     1
      ⋮


decodedInfo = struct with fields:
                        LAP: [24x1 double]
                 PacketType: 'DM1'
    LogicalTransportAddress: [3x1 double]
          HeaderControlBits: [3x1 double]
              PayloadLength: 10
                       LLID: [2x1 double]
              FlowIndicator: 1


pktStatus = logical
   1


crc = 16×1

     1
     1
     1
     1
     1
     1
     0
     0
     0
     0
      ⋮
```

Input Arguments

`waveform` — Synchronized time-domain Bluetooth BR/EDR waveform

complex-valued column vector

Synchronized time-domain Bluetooth BR/EDR waveform, specified as a complex-valued column vector.

Data Types: `double`
Complex Number Support: Yes

`rxConfig` — System configuration object

`bluetoothPHYConfig` object

System configuration object, specified as a `bluetoothPhyConfig` object.

Output Arguments

`bits` — Decoded payload bits

binary-valued column vector

Decoded payload bits, returned as a binary-valued column vector.

Data Types: `double`

`decodedInfo` — Decoded information

structure

Decoded information, returned as a structure containing these fields:

Field	Value	Description
`PacketType`	`'ID'`, `'NULL'`, `'POLL'`, `'FHS'`, `'HV1'`, `'HV2'`, `'HV3'`, `'DV'`, `'EV3'`, `'EV4'`, `'EV5'`, `'AUX1'`, `'DM3'`, `'DM1'`, `'DH1'`, `'DM5'`, `'DH3'`, `'DH5'`, `'2-DH1'`, `'2-DH3'`, `'2-DH5'`, `'2-DH1'`, `'2-DH3'`, `'2-DH5'`, `'2-EV3'`, `'2-EV5'`, `'3-EV3'`, or `'3-EV5'`	Type of received Bluetooth BR/EDR packet If the function does not detect any of the Bluetooth BR/EDR packets, this field returns an empty character vector.
`LAP`	24-bit column vector of type `double`.	Decoded lower address part (LAP) of the Bluetooth device address
`PayloadLength`	Scalar of type `double`	Number of payload bytes in the received Bluetooth BR/EDR packet
`LogicalTransportAddress`	3-bit vector of type `double`	Active destination Peripheral for a Bluetooth BR/EDR packet in a Central-to-Peripheral transmission slot
`HeaderControlBits`	3-bit vector of type `double`	Link control information containing flow control information (FLOW), acknowledgement for successfully receiving a Bluetooth BR/EDR packet payload (ARQN), and sequencing scheme for received packets (SEQN) bits
`LLID`	2-bit binary vector of type `double`	Logical link identifier. This field is applicable only if the value of `PacketType` field is one of these: `'DM1'`, `'DH1'`, `'DM3'`, `'DH3'`, `'DM5'`, `'DH5'`, `'AUX1'`, `'DV'`, `'2-DH1'`, `'2-DH3'`, `'2-DH5'`, `'3-DH1'`, `'3-DH3'`, or `'3-DH5'`.

Field	Value	Description
FlowIndicator	Scalar of type double	Control data flow indicator over logical channels. This field is applicable only if the value of PacketType field is one of these: 'DM1', 'DH1', 'DM3', 'DH3', 'DM5', 'DH5', 'AUX1', 'DV', '2-DH1', '2-DH3', '2-DH5', '3-DH1', '3-DH3', or '3-DH5'

Note From R2022a, this object uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Data Types: struct

pktValidStatus — Flag indicating validity of received Bluetooth BR/EDR packet
1 or true | 0 or false

Flag indicating validity of received Bluetooth BR/EDR packet, returned as 1 (true) or 0 (false). The validity is based on the Bluetooth BR/EDR packet header error check (HEC) and cyclic redundancy check (CRC). The value of this output is 1 or true only when HEC and CRC are enabled.

Dependencies

To enable this output argument, set the PacketType field value of the “decodedInfo” on page 1-0 output argument to any one of these: 'NULL', 'POLL', 'FHS', 'HV1', 'HV2', 'HV3', 'DV', 'EV3', 'EV4', 'EV5', 'AUX1', 'DM3', 'DM1', 'DH1', 'DM5', 'DH3', 'DH5', '2-DH1', '2-DH3', '2-DH5', '2-DH1', '2-DH3', '2-DH5', '2-EV3', '2-EV5', '3-EV3', or '3-EV5'.

Data Types: logical

decodedCRC — Decoded CRC
16-bit binary-valued column vector

Decoded CRC, specified as the CRC of the received Bluetooth BR/EDR packet.

Dependencies

To enable this output argument, set the PacketType field value of the decodedInfo output argument to any one of these: 'FHS', 'DV', 'EV3', 'EV4', 'EV5', 'DM3', 'DM1', 'DH1', 'DM5', 'DH3', 'DH5', '2-DH1', '2-DH3', '2-DH5', '2-DH1', '2-DH3', '2-DH5', '2-EV3', '2-EV5', '3-EV3', or '3-EV5'.

Data Types: double

Version History

Introduced in R2020a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

- Properties must be specified as coder.Constant().

See Also

Functions

`bluetoothWaveformGenerator` | `bleWaveformGenerator` | `bleIdealReceiver`

Objects

`bluetoothWaveformConfig` | `bluetoothPhyConfig` | `bluetoothWhiten`

Topics

“Bluetooth BR/EDR Waveform Reception Using SDR”

bluetoothPacketDuration

Compute Bluetooth BR/EDR or LE packet duration

Syntax

```
packetDuration = bluetoothPacketDuration(mode,packetType,payloadLength)
packetDuration = bluetoothPacketDuration( ___ ,cteLength)
[ ___ ,numBits] = bluetoothPacketDuration( ___ )
```

Description

`packetDuration = bluetoothPacketDuration(mode,packetType,payloadLength)` returns the duration, `packetDuration`, of the Bluetooth basic rate/enhanced data rate (BR/EDR) or low energy (LE) packet, based on the Bluetooth packet type `packetType` and length of the payload, `payloadLength`. The input, `packetType`, specifies the type of Bluetooth packet corresponding to the specified physical layer (PHY) transmission mode, `mode`.

`packetDuration = bluetoothPacketDuration( ___ ,cteLength)` specifies the length of the constant tone extension (CTE) for the `"ConnectionCTE"` and `"ConnectionlessCTE"` packet types in `"LE1M"` or `"LE2M"` PHY transmission mode.

`[ ___ ,numBits] = bluetoothPacketDuration( ___ )` additionally returns the number of bits in the Bluetooth BR/EDR or LE packet.

Examples

Compute Bluetooth BR Packet Duration

Set the PHY transmission mode to BR.

```
mode = "BR";
```

Specify the packet type and payload length of the Bluetooth BR packet.

```
packetType = "DH1";
payloadLen = 18;             % In bytes
```

Compute the Bluetooth BR packet duration.

```
packetDuration = bluetoothPacketDuration(mode,packetType,payloadLen) % In microseconds
```

```
packetDuration = 294
```

Compute Bluetooth LE Packet Duration

Set the PHY transmission mode to LE.

```
mode = "LE1M";
```

Specify the packet type and payload length of the Bluetooth LE packet.

```
packetType = "ConnectionCTE";
payloadLen = 120;             % In bytes
```

Set the length of the CTE. The function multiplies the specified value by 8 to get the duration in microseconds. For this example, specify 4 to set the length to 32 microseconds.

```
cteLength = 4;
```

Compute the Bluetooth LE packet duration.

```
packetDuration = bluetoothPacketDuration(mode,packetType,payloadLen,cteLength) % In microseconds
```

```
packetDuration = 1080
```

Compute Bluetooth EDR Packet Duration

Set the PHY transmission mode to EDR2M or EDR3M.

```
mode = "EDR2M";
```

Specify the packet type and payload length of the Bluetooth EDR packet.

```
packetType = "2-DH3";
payloadLen = 100;             % In bytes
```

Compute the packet duration and total number of bits in the Bluetooth EDR packet.

```
[packetDuration,numBits] = bluetoothPacketDuration(mode,packetType,payloadLen)
```

```
packetDuration = 560
```

```
numBits = 989
```

Input Arguments

`mode` — PHY transmission mode

`"BR"` | `"EDR2M"` | `"EDR3M"` | `"LE1M"` | `"LE2M"` | `"LE125K"` | `"LE500K"`

PHY transmission mode, specified as `"BR"`, `"EDR2M"`, `"EDR3M"`, `"LE1M"`, `"LE2M"`, `"LE125K"`, or `"LE500K"`. Specify Bluetooth PHY transmission mode based on the packet type you require.

Data Types: `char` | `string`

`packetType` — Bluetooth BR/EDR or LE packet type

`"ID"` | `"NULL"` | `"POLL"` | `"FHS"` | `"DM1"` | `...`

Bluetooth BR/EDR or LE packet type, specified as one of these values. When you set `packetType` input, you must set `mode` input to the corresponding value.

packetType Value	mode Value
• "ID" • "NULL" • "POLL" • "FHS" • "DM1" • "AUX1"	"BR", "EDR2M", or "EDR3M"
• "DH1" • "DM3" • "DH3" • "DM5" • "DH5" • "HV1" • "HV2" • "HV3" • "EV3" • "EV4" • "EV5" • "DV"	"BR"
• "2-DH1" • "2-DH3" • "2-DH5" • "2-EV3" • "2-EV5"	"EDR2M"
• "3-DH1" • "3-DH3" • "3-DH5" • "3-EV3" • "3-EV5"	"EDR3M"
• "ConnectionCTE" • "ConnectionlessCTE"	"LE1M" or "LE2M"
• "Disabled"	"LE1M", "LE2M", "LE125K", or "LE500K"

Data Types: char | string

payloadLength — Length of payload
nonnegative integer

Length of the payload, specified as a nonnegative integer. If you set the mode argument to "LE1M", "LE2M", "LE125K", or "LE500K", specify this input in the range [0, 256]. If you set mode argument to "BR", "EDR2M", or "EDR3M", this function sets this input based on the specified packetType value.

packetType Value	payloadLength Value
"ID", "NULL", or "POLL"	0
"FHS"	18
"HV1"	10
"HV2"	20
"DM1"	[0,17]
"DM3"	[0,121]
"DM5"	[0,224]
"DH1"	[0,121]
"DH3"	[0,183]
"DH5"	[0,339]
"EV3"	[1,30]
"EV4"	[1,120]
"EV5"	[1,180]
"DV"	[10,19]
"2-DH1"	[0,54]
"2-DH3"	[0,367]
"2-DH5"	[0,679]
"3-DH1"	[0,83]
"3-DH3"	[0,552]
"3-DH5"	[0,1021]
"2-EV3"	[1,60]
"2-EV5"	[1,360]
"3-EV3"	[1,90]
"3-EV5"	[1,540]
"AUX1"	[0,29]

Data Types: double

cteLength — Length of CTE
integer in the range [2, 20]

Length of the CTE, specified as an integer in the range [2, 20]. The function multiplies the specified value by 8 to get the duration in microseconds. You can specify this argument only when:

- mode is "LE1M" or "LE2M". packetType is "ConnectionCTE" or "ConnectionlessCTE".

Data Types: double

Output Arguments

packetDuration — Duration of Bluetooth BR/EDR or LE packet
integer

Duration of Bluetooth BR/EDR or LE packet, in microseconds, returned as an integer.

Data Types: `double`

`numBits` — Number of bits in Bluetooth BR/EDR or LE packet
integer

Number of bits in the Bluetooth BR/EDR or LE packet, returned as an integer. This output returns the total length of the Bluetooth BR/EDR or LE packet, in bits.

Data Types: `double`

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bluetoothWaveformGenerator` | `bleWaveformGenerator` | `bluetoothTestWaveform`

bluetoothRange

Estimate range between two Bluetooth BR/EDR or LE devices

Syntax

```
[range,pl,rxPower] = bluetoothRange(cfgRange)
```

Description

`[range,pl,rxPower] = bluetoothRange(cfgRange)` estimates the range between two Bluetooth basic rate/enhanced data rate (BR/EDR) or low energy (LE) devices. The function also returns the path loss, `pl`, and received signal power, `rxPower`, between two devices.

Examples

Estimate Range Between Two Bluetooth BR Devices in Office Environment

Create a default Bluetooth BR/EDR or LE range estimation configuration object.

```
cfgRange = bluetoothRangeConfig
```

```
cfgRange =
  bluetoothRangeConfig with properties:

                Environment: 'Outdoor'
             SignalPowerType: 'ReceiverSensitivity'
                        Mode: 'LE1M'
          ReceiverSensitivity: -94
                   LinkMargin: 15
             TransmitterPower: 0
       TransmitterAntennaGain: 0
          ReceiverAntennaGain: 0
         TransmitterCableLoss: 1.2500
            ReceiverCableLoss: 1.2500
     TransmitterAntennaHeight: 1
        ReceiverAntennaHeight: 1

   Read-only properties:
                FSPLDistance: 65.3645
               PathLossModel: 'TwoRayGroundReflection'
```

Set the physical layer (PHY) transmission mode and signal propagation environment to "BR" and "`Office`", respectively.

```
cfgRange.Mode = "BR";
cfgRange.Environment = "Office";
```

Specify the transmitter output power, transmitter cable loss, and receiver cable loss.

```
cfgRange.TransmitterPower = 20;         % In dBm
cfgRange.TransmitterCableLoss = 2.5;    % In dB
cfgRange.ReceiverCableLoss = 2;         % In dB
```

Estimate the range between two Bluetooth BR devices.

```
rangeBR = bluetoothRange(cfgRange)      % In meters
```

```
rangeBR = 1×2

   17.0455   22.8663
```

Estimate Distance Between Two Bluetooth Devices in Home Environment

Create a default Bluetooth BR/EDR or LE range estimation configuration object.

```
cfgRange = bluetoothRangeConfig
```

```
cfgRange =
  bluetoothRangeConfig with properties:

                Environment: 'Outdoor'
           SignalPowerType: 'ReceiverSensitivity'
                      Mode: 'LE1M'
       ReceiverSensitivity: -94
                 LinkMargin: 15
            TransmitterPower: 0
     TransmitterAntennaGain: 0
        ReceiverAntennaGain: 0
      TransmitterCableLoss: 1.2500
         ReceiverCableLoss: 1.2500
   TransmitterAntennaHeight: 1
      ReceiverAntennaHeight: 1

   Read-only properties:
             FSPLDistance: 65.3645
           PathLossModel: 'TwoRayGroundReflection'
```

Set the signal propagation environment to "Home" and the type of received signal power to "ReceivedSignalPower".

```
cfgRange.Environment = "Home";
cfgRange.SignalPowerType = "ReceivedSignalPower"
```

```
cfgRange =
  bluetoothRangeConfig with properties:

             Environment: 'Home'
          SignalPowerType: 'ReceivedSignalPower'
      ReceivedSignalPower: -79
         TransmitterPower: 0
    TransmitterAntennaGain: 0
       ReceiverAntennaGain: 0
     TransmitterCableLoss: 1.2500
```

```
         ReceiverCableLoss: 1.2500

   Read-only properties:
              FSPLDistance: 65.3645
             PathLossModel: 'NISTPAP02Task6'
```

Specify the received signal power and transmitter output power.

```
cfgRange.ReceivedSignalPower = -80;       % In dBm
cfgRange.TransmitterPower = 10            % In dBm
```

```
cfgRange =
  bluetoothRangeConfig with properties:

                Environment: 'Home'
           SignalPowerType: 'ReceivedSignalPower'
       ReceivedSignalPower: -80
          TransmitterPower: 10
    TransmitterAntennaGain: 0
       ReceiverAntennaGain: 0
      TransmitterCableLoss: 1.2500
         ReceiverCableLoss: 1.2500

   Read-only properties:
              FSPLDistance: 231.9220
             PathLossModel: 'NISTPAP02Task6'
```

Estimate the range between two Bluetooth devices.

```
range = bluetoothRange(cfgRange)
```

```
range = 1×2

   24.7467   32.5042
```

Estimate the distance between two Bluetooth devices by considering a random value from the estimated range.

```
distanceValues = min(range):max(range);
index = randi(length(distanceValues),1);
distance = distanceValues(index)          % In meters
```

```
distance = 30.7467
```

Input Arguments

`cfgRange` — Bluetooth BR/EDR or LE range estimation configuration parameters

`bluetoothRangeConfig` object

Bluetooth BR/EDR or LE range estimation configuration parameters, specified as a `bluetoothRangeConfig` object.

Output Arguments

`range` — Range between two Bluetooth BR/EDR or LE devices
vector

Range between two Bluetooth BR/EDR or LE devices, returned as a 1-by-2 vector. Units are in meters.

Data Types: `double`

`pl` — Path loss between two Bluetooth BR/EDR or LE devices
scalar

Path loss between two Bluetooth BR/EDR or LE devices, returned as a scalar. Units are in dB.

Data Types: `double`

`rxPower` — Received signal power between two Bluetooth BR/EDR or LE devices
scalar

Received signal power between two Bluetooth BR/EDR or LE devices, returned as a scalar. Units are in dBm.

Data Types: `double`

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 12, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

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

See Also

Functions
`bleAngleEstimate` | `blePositionEstimate`

Objects
`bluetoothRangeConfig`

Topics
“Bluetooth Location and Direction Finding”
“Parameterize Bluetooth LE Direction Finding Features”

“Estimate Bluetooth LE Node Position”
“Bluetooth LE Positioning by Using Direction Finding”
“Bluetooth LE Direction Finding for Tracking Node Position”

bluetoothTestWaveform

Generate Bluetooth BR/EDR or LE test waveform

Syntax

```
testWaveform = bluetoothTestWaveform(cfgTestWaveform)
```

Description

`testWaveform = bluetoothTestWaveform(cfgTestWaveform)` generates the Bluetooth basic rate/enhanced data rate (BR/EDR) or low energy (LE) test waveform.

Examples

Generate Bluetooth BR/EDR and LE Test Waveform

This example shows you how to generate a Bluetooth LE or BR/EDR test waveform for these physical layer (PHY) transmission modes.

- Bluetooth LE: LE1M, LE2M, LE125K, LE500K
- Bluetooth BR/EDR: BR, EDR2M, and EDR3M

Generate two Bluetooth test waveforms: one with the PHY transmission mode as LE1M and one with PHY transmission mode as BR.

Generate Bluetooth Test Waveform with LE1M Mode

Create a default Bluetooth BR/EDR and LE test waveform configuration object.

```
cfgLETestWaveform = bluetoothTestWaveformConfig
```

```
cfgLETestWaveform =
  bluetoothTestWaveformConfig with properties:

                Mode: 'LE1M'
       PayloadLength: 255
          PacketType: 'Disabled'
         PayloadType: 0
    SamplesPerSymbol: 8
```

Generate a Bluetooth test waveform with the LE1M PHY transmission mode.

```
txLEWaveform = bluetoothTestWaveform(cfgLETestWaveform);
```

Generate Bluetooth Test Waveform with BR Mode

Create a default Bluetooth BR/EDR and LE test waveform configuration object.

```
cfgBRTestWaveform = bluetoothTestWaveformConfig;
```

Set the PHY transmission mode to BR.

```
cfgBRTestWaveform.Mode = "BR"
```

```
cfgBRTestWaveform =
  bluetoothTestWaveformConfig with properties:

                  Mode: 'BR'
            WhitenStatus: 'Off'
       ModulationIndex: 0.3200
            PacketType: 'DH1'
           PayloadType: 0
      SamplesPerSymbol: 8

   Read-only properties:
    FixedPayloadLength: 27
```

Generate a Bluetooth test waveform with the BR PHY transmission mode.

```
txBRWaveform = bluetoothTestWaveform(cfgBRTestWaveform);
```

Input Arguments

`cfgTestWaveform` — Bluetooth BR/EDR or LE test waveform configuration

`bluetoothTestWaveformConfig` object

Bluetooth BR/EDR or LE test waveform configuration, specified as a `bluetoothTestWaveformConfig` object.

Output Arguments

`testWaveform` — Bluetooth BR/EDR or LE test waveform

complex-valued column vector

Bluetooth BR/EDR or LE test waveform, returned as a complex-valued column vector of length *Ns*, where *Ns* is the number of time-domain samples.

Data Types: `double`

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleWaveformGenerator` | `bluetoothWaveformGenerator` | `bleIdealReceiver` | `bluetoothIdealReceiver` | `bleCTEIQSample`

Objects

`bluetoothRFPHYTestConfig` | `bluetoothTestWaveformConfig` | `bluetoothWaveformConfig` | `bluetoothPhyConfig`

Topics

“Bluetooth LE Output Power and In-Band Emissions Tests”
“Bluetooth LE Blocking, Intermodulation and Carrier-to-Interference Performance Tests”
“Bluetooth LE Modulation Characteristics, Carrier Frequency Offset and Drift Tests”
“Bluetooth EDR RF-PHY Transmitter Tests for Modulation Accuracy and Carrier Frequency Stability”
“Bluetooth BR RF-PHY Transmitter Tests for Modulation Characteristics, Carrier Frequency Offset, and Drift”
“Bluetooth LE IQ samples Coherency and Dynamic Range Tests”
“Bluetooth BR/EDR Power and Spectrum Tests”

bluetoothWaveformGenerator

Generate Bluetooth BR/EDR PHY waveform

Syntax

```
waveform = bluetoothWaveformGenerator(data,cfgFormat)
```

Description

`waveform = bluetoothWaveformGenerator(data,cfgFormat)` generates `waveform`, a multipacket time-domain Bluetooth BR/EDR waveform, for input information bits, `data`, and a given format configuration, `cfgFormat`.

Examples

Generate Multipacket Bluetooth BR/EDR Waveform with HV1 Packets

Specify the number of HV1 packets.

```
numPackets = 10;
```

Create a default Bluetooth BR/EDR waveform configuration object. Specify the packet type as HV1.

```
cfgWaveform = bluetoothWaveformConfig;
cfgWaveform.PacketType = 'HV1';
```

Create a random input bit vector containing concatenated payloads.

```
numBits = getPayloadLength(cfgWaveform)*8*numPackets; % Byte to bit conversion
dataBits = randi([0 1],numBits,1);
```

Set the symbol rate.

```
symbolRate = 1e6; % In MHz
```

Generate the Bluetooth BR/EDR waveform.

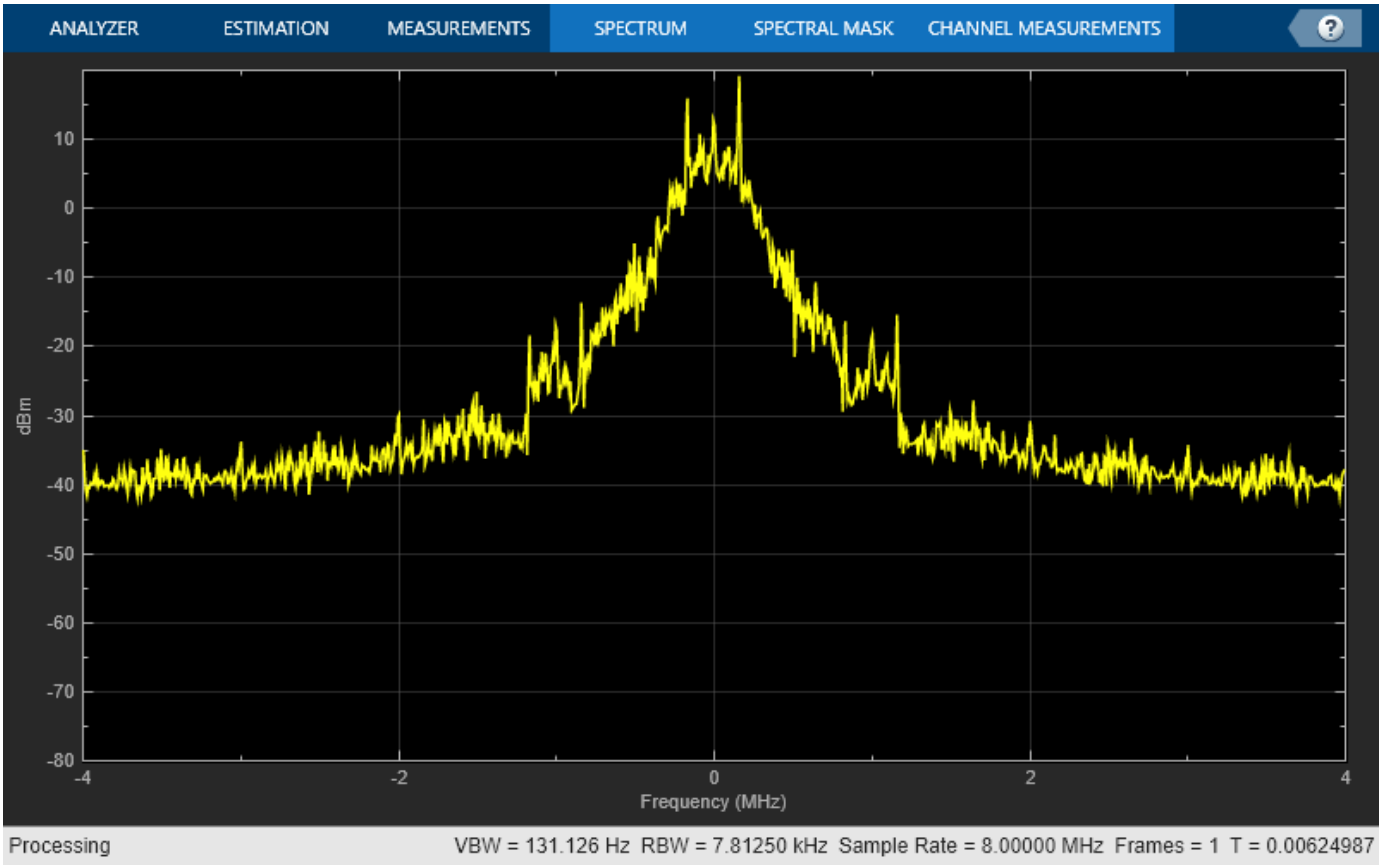
```
waveform = bluetoothWaveformGenerator(dataBits,cfgWaveform);
```

Create a spectrum analyzer System object to display the frequency spectrum of the generated Bluetooth BR/EDR waveform. Set the sample rate of the frequency spectrum.

```
scope = spectrumAnalyzer;
scope.SampleRate = cfgWaveform.SamplesPerSymbol*symbolRate;
```

Plot the Bluetooth BR/EDR waveform.

```
scope(waveform);
```



Generate Bluetooth BR/EDR Waveform for Single Enhanced Data Rate Packet

Create a default Bluetooth BR/EDR waveform configuration object.

```
cfgWaveform = bluetoothWaveformConfig;
```

To generate enhanced data rate packet, 2-EV3, specify the packet type as EV3 and the PHY transmission mode as EDRM2.

```
cfgWaveform.PacketType = 'EV3';
cfgWaveform.Mode = 'EDR2M';
```

Create a random input bit vector to generate the payload for a single packet.

```
numBits = getPayloadLength(cfgWaveform)*8;   % Byte to bit conversion
dataBits = randi([0 1],numBits,1);
```

Generate the Bluetooth BR/EDR waveform.

```
txWaveform = bluetoothWaveformGenerator(dataBits,cfgWaveform);
```

Input Arguments

`data` — Input information bits

binary-valued column vector

Input information bits, specified as a binary-valued column vector representing multiple concatenated payloads. Specify this input as an exact multiple of the payload length derived from the `getPayloadLength` object function of the `bluetoothWaveformConfig` object.

Data Types: `double`

`cfgFormat` — Format configuration object

`bluetoothWaveformConfig` object

Format configuration object, specified as a `bluetoothWaveformConfig` object.

Output Arguments

`waveform` — Generated time-domain Bluetooth BR/EDR waveform

complex-valued column vector

Generated time-domain Bluetooth BR/EDR waveform, returned as a complex-valued column vector containing the generated Bluetooth BR/EDR waveform. The function appends this value with zero samples to accommodate a packet-specific slot duration.

Data Types: `double`

Version History

Introduced in R2020a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Properties must be specified as coder.Constant().

See Also

Functions

`bluetoothIdealReceiver` | `bleWaveformGenerator` | `bleIdealReceiver`

Objects
`bluetoothWaveformConfig` | `bluetoothPhyConfig` | `bluetoothWhiten`

Topics
“Bluetooth Packet Structure”
“Configure Bluetooth BR/EDR Channel with WLAN Interference and Pass the Waveform Through Channel”
“Bluetooth BR/EDR Waveform Generation and Transmission Using SDR”
“Bluetooth BR/EDR Waveform Reception Using SDR”

Objects

bleAngleEstimateConfig

Bluetooth LE angle estimation configuration parameters

Description

The `bleAngleEstimateConfig` object parameterizes the `bleAngleEstimate` function for estimating the Bluetooth low energy (LE) angle of arrival (AoA) or angle of departure (AoD).

Creation

Syntax

```
cfgAngle = bleAngleEstimateConfig
cfgAngle = bleAngleEstimateConfig(Name,Value)
```

Description

`cfgAngle = bleAngleEstimateConfig` creates a default Bluetooth LE angle estimation configuration object.

`cfgAngle = bleAngleEstimateConfig(Name,Value)` sets properties on page 2-2 by using one or more name-value pairs. Enclose each property name in quotes. For example, `bleAngleEstimateConfig('SlotDuration',1)` sets the switch and sample slot duration to 1 µs.

Properties

`ArraySize` — Size of array
`4` (default) | positive integer | two-element row vector of positive integers

Size of the array, specified as a positive integer or a two-element row vector of positive integers.

- Setting this property to a positive integer specifies a uniform linear array (ULA) antenna array design with array elements on the *y*-axis.
- Setting this property to a vector specifies a uniform rectangular array (URA) antenna array design. The vector must be of the form [*r c*], where *r* and *c* denote the number of row elements and column elements in the antenna array, respectively. In this case, the row elements and column elements are present along *y*-axis and *z*-axis, respectively.

Dependencies

To enable this property, set the EnableCustomArray property to `0` or `false`.

Data Types: `double`

`ElementSpacing` — Normalized element spacing with respect to signal wavelength
`0.5` (default) | positive real number | two-element row vector of positive real numbers

Normalized element spacing with respect to signal wavelength, specified as a positive real number or a two-element row vector of positive real numbers.

- Setting this property to a positive real number, specifies a ULA antenna array design with array elements on the *y*-axis.
- Setting this property to a vector, specifies a URA antenna array design. The vector must be of the form [*sr sc*], where *sr* and *sc* denote the spacing between row and column elements of the antenna array, respectively. In this case, the rows and columns are present along *y*-axis and *z*-axis, respectively.

Dependencies

To enable this property, set the EnableCustomArray property to `0` or `false`.

Data Types: `double`

`EnableCustomArray` — Flag to enable or disable custom array

`0` or `false` (default) | `1` or `true`

Flag to enable or disable custom array, specified as `0` (`false`) or `1` (`true`).

Data Types: `logical`

`ElementPosition` — Normalized element positions with respect to wavelength

`[0 0.5 1 1.5; 0 0.5 1 1.5; 0 0.5 1 1.5]` (default) | 3-by-*N* matrix

Normalized element positions with respect to wavelength, specified as a 3-by-*N* matrix, where *N* is the number of elements in the custom array. Each column of the matrix specifies the element position in the form [*x*; *y*; *z*] in the local coordinate system. The first column of the matrix must be [`0`; `0`; `0`].

Dependencies

To enable this property, set the EnableCustomArray property to `1` or `true`.

Data Types: `double`

`SlotDuration` — Switch and sample slot duration

`2` (default) | `1`

Switch and sample slot duration, specified as `1` or `2`. Units are in microseconds.

Data Types: `double`

`SwitchingPattern` — Antenna switching pattern

`[1 2 3 4]` (default) | *M*-element row vector

Antenna switching pattern, specified as an *M*-element row vector, where *M* must be in the range [2, 74/“SlotDuration” on page 2-0 +1].

Data Types: `double`

Object Functions

Specific to This Object

getElementPosition Return positions of array elements

getNumElements Return number of elements in array

Examples

Create Bluetooth LE Angle Estimation Configuration Object With Four-Element ULA

Create a default Bluetooth LE angle estimation configuration object.

```
cfgAngle = bleAngleEstimateConfig;
```

Specify a ULA antenna array design by setting the antenna array size of the configuration object to 4.

```
cfgAngle.ArraySize = 4
```

```
cfgAngle =
  bleAngleEstimateConfig with properties:

            ArraySize: 4
       ElementSpacing: 0.5000
    EnableCustomArray: 0
         SlotDuration: 2
     SwitchingPattern: [1 2 3 4]
```

Create Bluetooth LE Angle Estimation Configuration Object with 4-by-4 URA

Create a default Bluetooth LE angle estimation configuration object.

```
cfgAngle = bleAngleEstimateConfig
```

```
cfgAngle =
  bleAngleEstimateConfig with properties:

            ArraySize: 4
       ElementSpacing: 0.5000
    EnableCustomArray: 0
         SlotDuration: 2
     SwitchingPattern: [1 2 3 4]
```

Specify a URA antenna array design by setting the antenna array size of the configuration object to [4 4].

```
cfgAngle.ArraySize = [4 4];
```

Set the row element spacing and column element spacing to `0.4` and `0.3`, respectively.

```
cfgAngle.ElementSpacing = [0.4 0.3];
```

Set the value of antenna switching pattern.

```
cfgAngle.SwitchingPattern = 1:16
```

```
cfgAngle =
  bleAngleEstimateConfig with properties:
```

```
            ArraySize: [4 4]
       ElementSpacing: [0.4000 0.3000]
    EnableCustomArray: 0
         SlotDuration: 2
     SwitchingPattern: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]
```

Create Bluetooth LE Angle Estimation Configuration Object With Uniform Circular Array (UCA)

Create a default Bluetooth LE angle estimation configuration object.

```
cfgAngle = bleAngleEstimateConfig
```

```
cfgAngle =
  bleAngleEstimateConfig with properties:

            ArraySize: 4
       ElementSpacing: 0.5000
    EnableCustomArray: 0
         SlotDuration: 2
     SwitchingPattern: [1 2 3 4]
```

Enable custom antenna array support.

```
cfgAngle.EnableCustomArray = 1;
```

Specify the normalized element positions with respect to wavelength.

```
cfgAngle.ElementPosition = [0 0 0 0;0 0.5 1 0.5;0 -0.5 0 0.5];
```

Specify the antenna switching pattern.

```
cfgAngle.SwitchingPattern = 1:4
```

```
cfgAngle =
  bleAngleEstimateConfig with properties:

    EnableCustomArray: 1
      ElementPosition: [3x4 double]
         SlotDuration: 2
     SwitchingPattern: [1 2 3 4]
```

Version History

Introduced in R2020b

Includes support for custom antenna arrays

The object includes two new properties, EnableCustomArray and ElementPosition, that support custom antenna arrays for AoA and AoD estimation.

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.1. https://www.bluetooth.com/.

[3] Wooley, Martin. *Bluetooth Direction Finding: A Technical Overview*. Bluetooth Special Interest Group (SIG), Accessed April 6, 2020, https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleAngleEstimate` | `bleWaveformGenerator` | `bleIdealReceiver`

Topics

“Bluetooth Packet Structure”
“Bluetooth Location and Direction Finding”
“Parameterize Bluetooth LE Direction Finding Features”
“Bluetooth LE Positioning by Using Direction Finding”
“Bluetooth LE Direction Finding for Tracking Node Position”

bleATTPDUConfig

Bluetooth LE ATT PDU configuration parameters

Description

The `bleATTPDUConfig` object parameterizes the `bleATTPDU` function to generate a Bluetooth low energy (LE) attribute protocol data unit (ATT PDU).

Creation

Syntax

```
cfgATT = bleATTPDUConfig
cfgATT = bleATTPDUConfig(Name,Value)
```

Description

`cfgATT = bleATTPDUConfig` creates a default Bluetooth LE ATT PDU configuration object.

`cfgATT = bleATTPDUConfig(Name,Value)` sets properties on page 2-7 using one or more name-value pairs. Enclose each property name in quotes. For example, `('Opcode','Error response')` sets the operation code to `'Error response'`.

Properties

Note For more information about Bluetooth LE ATT PDU properties and their respective values, see Volume 3, Part F, Sections 3.3 and 3.4 of the Bluetooth Core Specification [2].

Opcode — Bluetooth LE ATT PDU operation code

`'Read request'` (default) | `'MTU request'` | `'Information request'` | `...`

Bluetooth LE ATT PDU operation code, specified as one of these values.

- `'MTU request'`
- `'MTU response'`
- `'Error response'`
- `'Information request'`
- `'Find by type value request'`
- `'Read by type request'`
- `'Read request'`
- `'Read response'`
- `'Read blob request'`

- 'Read blob response'
- 'Read by group type request'
- 'Write request'
- 'Write response'
- 'Write command'
- 'Prepare write request'
- 'Prepare write response'
- 'Execute write request'
- 'Execute write response'
- 'Handle value notification'
- 'Handle value indication'
- 'Handle value confirmation'
- 'Information response'
- 'Find by type value response'
- 'Read by type response'
- 'Read by group type response'

Data Types: char | string

RequestedOpcode — Opcode of request Bluetooth LE ATT PDU
'Read request' (default) | character vector | string scalar

Opcode of request Bluetooth LE ATT PDU, specified as one of the values in this list. Each valid value describes a request Bluetooth LE ATT PDU (from a peer device) that caused an error.

- 'MTU request'
- 'Information request'
- 'Find by type value request'
- 'Read by type request'
- 'Read request'
- 'Read blob request'
- 'Read by group type request'
- 'Write request'
- 'Prepare to write request'
- 'Execute write request'

Data Types: char | string

Format — Format of information data field
'16 bit' (default) | '128 bit'

Format of information data field, specified as '16 bit' or '128 bit'. This value specifies the format of the information data element in the PDU with Opcode 'Information Response'.

Data Types: char | string

`AttributeHandle` — Handle value of attribute
`'0001'` (default) | character vector of 2-octet hexadecimal

Handle value of attribute, specified as the character vector of a 2-octet hexadecimal value in the range [0x0001, 0xFFFF]. This value is a unique identifier. The server dynamically assigns this value.

Data Types: `char` | `string`

`ErrorMessage` — Error message corresponding to request Bluetooth LE ATT PDU
`'Invalid handle'` (default) | `'Invalid handle'` | `'Read not permitted'` | `...`

Error message corresponding to request Bluetooth LE ATT PDU, specified as one of the values in this list. Each value indicates the cause of an error corresponding to the request Bluetooth LE ATT PDU from a peer device.

- `'Invalid handle'`
- `'Read not permitted'`
- `'Write not permitted'`
- `'Invalid PDU'`
- `'Insufficient authentication'`
- `'Request not supported'`
- `'Invalid offset'`
- `'Insufficient authorization'`
- `'Prepare queue full'`
- `'Attribute not found'`
- `'Attribute not long'`
- `'Insufficient encryption key size'`
- `'Invalid attribute value length'`
- `'Unlikely error'`
- `'Insufficient encryption'`
- `'Unsupported group type'`
- `'Insufficient resources'`

Data Types: `char` | `string`

`MaxTransmissionUnit` — Maximum size of Bluetooth LE ATT PDU
`23` (default) | integer in the range [23, 65,535]

Maximum size of Bluetooth LE ATT PDU, specified as an integer in the range [23, 65,535]. This value sets the maximum size of the Bluetooth LE ATT PDU in bytes that a client or a server can receive.

Data Types: `double`

`StartHandle` — Starting handle of handle range
`'0001'` (default) | character vector of 2-octet hexadecimal value

Starting handle of handle range, specified as a 2-octet hexadecimal value in the range [0x0001,0xFFFF]. This value indicates the handle value of a service, characteristic declaration, or the starting handle of a handle range. This value must be less than the EndHandle property value.

Data Types: `char` | `string`

`EndHandle` — Ending handle of handle range
`'FFFF'` (default) | character vector of 2-octet hexadecimal value

Ending handle of handle range, specified as a 2-octet hexadecimal value in the range [0x0001,0xFFFF]. This value sets the end handle value of a service declaration, characteristic declaration, or the ending handle of a handle range. This value must be greater than the StartHandle value.

Data Types: `char` | `string`

`AttributeType` — Type of attribute
`'2800' (Primary service)` (default) | 4-element or 32-element character vector | 2-octet or 16-octet string scalar

Type of attribute, specified as a 4-element or 32-element character vector or a string scalar denoting a 2-octet or 16-octet hexadecimal value.

Data Types: `char` | `string`

`AttributeValue` — Value of attribute
`' '` (default) | character vector | string scalar | numeric vector of elements in the range [0, 255] | *n*-by-2 character array of maximum length `131068`

Value of attribute, specified as one of these values:

- Character vector — This vector represent octets in hexadecimal format.
- String scalar — This scalar represent octets in hexadecimal format.
- Numeric vector of elements in the range [0, 255] — This vector represent octets in decimal format. The maximum length of the numeric vector is `65534`.
- n-by-2 character array — Each row represent an octet in hexadecimal format. The maximum length of the character array is `131068`.

This property specifies the value of an attribute to be stored in or read from the attribute database. Specify this value in least significant bit (LSB) first format.

Data Types: `char` | `string` | `double`

`Offset` — Offset of next octet to be read
`0` (default) | integer in the range [0, 65,565]

Offset of next octet to be read, specified as an integer in the range [0, 65,535]. To identify an attribute value offset in the attribute database, use this value in the Bluetooth LE ATT PDUs with opcodes `'Read blob request'`, `'Prepare write request'`, and `'Prepare write response'`.

Data Types: `double`

`ExecuteWrite` — Execute write flag
`'Cancel all prepared writes'` (default) | `'Write all pending requests'`

Execute write flag, specified as `'Cancel all prepared writes'` or `'Write all pending requests'`. The object performs the discard or write action by setting this value.

Data Types: `char` | `string`

Examples

Create Bluetooth LE ATT PDU Configuration Objects

Create two unique Bluetooth LE ATT PDU configuration objects: one of type `'Read by type request'` and the other of type `'Error response'`.

Create a default Bluetooth LE ATT PDU configuration object.

```
cfgATT = bleATTPDUConfig
```

```
cfgATT =
  bleATTPDUConfig with properties:

             Opcode: 'Read request'
    AttributeHandle: '0001'
```

Set the Bluetooth LE ATT PDU opcode as `'Read by type request'`.

```
cfgATT.Opcode = 'Read by type request'
```

```
cfgATT =
  bleATTPDUConfig with properties:

           Opcode: 'Read by type request'
      StartHandle: '0001'
        EndHandle: 'FFFF'
    AttributeType: '2800'
```

Create another Bluetooth LE ATT PDU configuration object, specifying the opcode as `'Error response'`.

```
cfgATT = bleATTPDUConfig('Opcode','Error response')
```

```
cfgATT =
  bleATTPDUConfig with properties:

             Opcode: 'Error response'
    RequestedOpcode: 'Read request'
    AttributeHandle: '0001'
       ErrorMessage: 'Invalid handle'
```

End-to-End Workflow of Bluetooth LE ATT PDU

Create a Bluetooth LE ATT PDU configuration object. Set the opcode to `'Read by type request'`.

```
cfgATTPDUTx = bleATTPDUConfig;
cfgATTPDUTx.Opcode = 'Read by type request'
```

```
cfgATTPDUTx =
  bleATTPDUConfig with properties:
```

```
           Opcode: 'Read by type request'
      StartHandle: '0001'
        EndHandle: 'FFFF'
    AttributeType: '2800'
```

Generate a Bluetooth LE ATT PDU from the corresponding configuration object.

```
attPDU = bleATTPDU(cfgATTPDUTx);
```

Decode the generated Bluetooth LE ATT PDU. The returned status indicates decoding is successful.

```
[status,cfgATTPDURx] = bleATTPDUDecode(attPDU)
```

```
status =
  blePacketDecodeStatus enumeration

    Success


cfgATTPDURx =
  bleATTPDUConfig with properties:

           Opcode: 'Read by type request'
      StartHandle: '0001'
        EndHandle: 'FFFF'
    AttributeType: '2800'
```

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleATTPDU` | `bleATTPDUDecode`

Topics

“Bluetooth Packet Structure”

“Generate and Decode Bluetooth Protocol Data Units”

bleGAPDataBlockConfig

Bluetooth LE GAP data block configuration parameters

Description

The `bleGAPDataBlockConfig` parameterizes the `bleGAPDataBlock` function to generate a Bluetooth low energy (LE) generic access profile (GAP) data block of the type advertising data (AD) or scan response data (SRD).

Creation

Syntax

```
cfgGAP = bleGAPDataBlockConfig
cfgGAP = bleGAPDataBlockConfig(Name,Value)
```

Description

`cfgGAP = bleGAPDataBlockConfig` creates a default configuration object, `cfgGAP`, for a Bluetooth LE GAP data block of the type AD or SRD.

`cfgGAP = bleGAPDataBlockConfig(Name,Value)` sets properties on page 2-14 using one or more name-value pairs. Enclose each property name in quotes. For example, `('AdvertisingDataTypes','Tx power level')` sets the block data advertising data type to `'Tx power level'`.

Properties

Note For more information about Bluetooth LE GAP data block properties and their respective values, see Volume 3, Part C, Section 4 of the Bluetooth Core Specification [2].

`AdvertisingDataTypes` — Block data advertising data types

`'Flags'` (default) | `'UUIDs'` | `'Local name'` | `'Tx power level'` | `'Connection interval range'` | `'Advertising interval'`

Block data advertising data types, specified as a character vector, a string scalar or a cell array, containing the list of advertising data types for Bluetooth LE GAP data block. Specify this property as one of these values:

- `'Flags'`
- `'UUIDs'`
- `'Local name'`
- `'Tx power level'`

- `'Connection interval range'`
- `'Advertising interval'`

Data Types: `char` | `string` | `cell`

`LEDiscoverability` — LE discoverable mode
`'General'` (default) | `'None'` | `'Limited'` | `'Limited and general'`

LE discoverable mode, specified as a character vector or a string scalar, describing the LE discoverable mode of the device. Specify this property as one of these values:

- `'None'`
- `'General'`
- `'Limited'`
- `'Limited and general'`

Data Types: `char` | `string`

`BREDR` — Flag to enable basic rate (BR) or enhanced data rate (EDR) support
`0` or `false` | `1` or `true`

Flag to enable basic rate (BR) or enhanced data rate (EDR) support, specified as `1` (`true`) or `0` (`false`). A `1` (`true`) value indicates that the object supports BR or EDR mode.

Dependencies

To enable this property, set the `AdvertisingDataTypes` property to `'Flags'`.

Data Types: `logical`

`LE` — Simultaneous LE and BR or EDR support
`'None'` (default) | `'Host'` | `'Controller'` | `'Host and controller'`

Simultaneous LE and BR/EDR support, specified as a character vector or a string scalar. Specify this property as one of these values:

- `'None'` — Simultaneous LE and BR or EDR support is disabled
- `'Host'` — Simultaneous LE and BR or EDR support is enabled at the Host
- `'Controller'` — Simultaneous LE and BR or EDR support is enabled at the Controller
- `'Host and controller'` — Simultaneous LE and BR or EDR support is enabled at the Host and Controller

Dependencies

To enable this property, set the BREDR property to `true`.

Data Types: `char` | `string`

`LocalNameShortening` — Flag to enable shortening of local name
`0` or `false` | `1` or `true`

Flag to enable shortening of local name, specified as `1` (`true`) or `0` (`false`).

Data Types: `logical`

`LocalName` — UTF-8 encoded user-friendly descriptive name
`'Bluetooth'` (default) | character vector or a string scalar consisting of UTF-8 characters

UTF-8 encoded user-friendly descriptive name, specified as a character vector or a string scalar consisting of UTF-8 characters. This property specifies the local name that the object assigns to the device.

Data Types: `char` | `string`

`IdentifierType` — Type of 16-bit service or service class identifiers
`'Incomplete'` (default) | `'Complete'`

Type of 16-bit service or service class identifiers, specified as `'Incomplete'` or `'Complete'`. If you set this value to `'Incomplete'`, the "Identifiers" on page 2-0 property is incomplete.

Data Types: `char` | `string`

`Identifiers` — List of 16-bit service or service class identifiers
`'' (empty)` (default) | *n*-by-4 character array

List of 16-bit service or service class identifiers, specified as an *n*-by-4 character array. The value of *n* must be in the range [0, 127]. Each row in the *n*-by-4 character array is represented as a 4-element character vector or a string scalar denoting a 2-octet (16-bit) hexadecimal value of a service or service class universally unique identifier (UUID). These UUIDs are assigned by the Bluetooth Special Interest Group (SIG).

Data Types: `char`

`AdvertisingInterval` — Advertising interval
`32` (default) | integer in the range [32, 65,535]

Advertising interval, specified as an integer in the range [32, 65,535]. This property denotes the interval between the start of two consecutive advertising events. Incremental units are in 0.625 ms steps, so the resultant range for [32, 65,535] is [20, 40.959375].

Data Types: `double`

`TxPowerLevel` — Packet transmit power level
`0 (dBm)` (default) | integer in the range [–127, 127]

Packet transmit power level, specified as an integer in the range [–127, 127]. This property calculates the pathloss as *pathloss = Tx Power Level – RSSI*, where *RSSI* is the received signal strength indicator.

Data Types: `double`

`ConnectionIntervalRange` — Connection interval range
`[6,3200]` (default) | 2-element numeric vector [MIN, MAX]

Connection interval range, specified as a 2-element numeric vector [*MIN*, *MAX*], where *MIN* and *MAX* values must be in the range [6, 3200]. *MIN* and *MAX* specify the minimum and maximum value for the connection interval, respectively. *MIN* must be less than or equal to *MAX*. Incremental units are 1.25 ms steps, so that the resultant range for [6, 3200] is [7.5, 4.0].

Data Types: `double`

Examples

Create Bluetooth LE GAP AD Block Configuration Objects

Create two unique Bluetooth LE GAP AD configuration objects: one with AD types `'Flags'` and `'Tx power level'` and the other with AD type `'Flags'` and simultaneous LE and BR or EDR support at the host.

Create a default Bluetooth LE GAP AD block configuration object. Set the values of AD types to `'Flags'` and `'Tx power level'`, LE discoverability to `'Limited'` and Tx power level to `45`.

```
cfgGAP = bleGAPDataBlockConfig;
cfgGAP.AdvertisingDataTypes = {'Flags';'Tx power level'};
cfgGAP.LEDiscoverability = 'Limited';
cfgGAP.TxPowerLevel = 45
```

```
cfgGAP =
  bleGAPDataBlockConfig with properties:

    AdvertisingDataTypes: {2x1 cell}
       LEDiscoverability: 'Limited'
                   BREDR: 0
            TxPowerLevel: 45
```

Create another default Bluetooth LE GAP AD block configuration object, this time with AD type `'Flags'` and having simultaneous support for LE and BR/EDR at the Host. Set the values of LE discoverability to `'Limited'`. Enable BR or EDR support,

```
cfgGAP = bleGAPDataBlockConfig;
cfgGAP.AdvertisingDataTypes = {'Flags'};
cfgGAP.LEDiscoverability = 'Limited and general';
cfgGAP.BREDR = true;
cfgGAP.LE = 'Host'
```

```
cfgGAP =
  bleGAPDataBlockConfig with properties:

    AdvertisingDataTypes: {'Flags'}
       LEDiscoverability: 'Limited and general'
                   BREDR: 1
                      LE: 'Host'
```

Create Bluetooth LE GAP AD Block Configuration Object Using Name-Value Arguments

Create a configuration object for a Bluetooth LE GAP AD block by using name-value pairs. Set the values of AD types to `'Advertising interval'` and `'Local name'`, advertising interval to `48`, local name to `'MathWorks'` and local name shortening to `true`.

```
cfgGAP = bleGAPDataBlockConfig('AdvertisingDataTypes', ...
    {'Advertising interval', ...
    'Local name'});
cfgGAP.AdvertisingInterval = 48;
```

```
cfgGAP.LocalName = 'MathWorks';
cfgGAP.LocalNameShortening = true
```

```
cfgGAP =
  bleGAPDataBlockConfig with properties:

    AdvertisingDataTypes: {2x1 cell}
               LocalName: 'MathWorks'
     LocalNameShortening: 1
     AdvertisingInterval: 48
```

End-to-End Workflow of Bluetooth LE GAP AD Blocks

Create a Bluetooth LE GAP AD block configuration object by using name-value pairs. Set the values of AD types to `'Advertising interval'` and `'Local name'`, advertising interval to `48`, local name as `'MathWorks'`, and local name shortening to `true`.

```
cfgGAPTx = bleGAPDataBlockConfig('AdvertisingDataTypes',{'Advertising interval','Local name'});
cfgGAPTx.AdvertisingInterval = 48;
cfgGAPTx.LocalName = 'MathWorks';
cfgGAPTx.LocalNameShortening = true
```

```
cfgGAPTx =
  bleGAPDataBlockConfig with properties:

    AdvertisingDataTypes: {2x1 cell}
               LocalName: 'MathWorks'
     LocalNameShortening: 1
     AdvertisingInterval: 48
```

Generate a Bluetooth LE GAP AD block.

```
dataBlock = bleGAPDataBlock(cfgGAPTx);
```

Decode the generated Bluetooth LE GAP AD block. The returned status indicates decoding was successful.

```
[status,cfgGAPRx] = bleGAPDataBlockDecode(dataBlock)
```

```
status =
  blePacketDecodeStatus enumeration

    Success


cfgGAPRx =
  bleGAPDataBlockConfig with properties:

    AdvertisingDataTypes: {2x1 cell}
               LocalName: 'MathWorks'
     LocalNameShortening: 1
     AdvertisingInterval: 48
```

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "Supplement to the Bluetooth Core Specification." CSS Version 7 (2016).

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleGAPDataBlock` | `bleGAPDataBlockDecode`

Topics

“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”

bleL2CAPFrameConfig

Bluetooth LE L2CAP frame configuration parameters

Description

The `bleL2CAPFrameConfig` parameterizes the `bleL2CAPFrame` function to generate a Bluetooth low energy (LE) logical link control and adaptation protocol (L2CAP) signaling frame or data frame.

Creation

Syntax

```
cfgL2CAP = bleL2CAPFrameConfig
cfgL2CAP = bleL2CAPFrameConfig(Name,Value)
```

Description

`cfgL2CAP = bleL2CAPFrameConfig` creates a default configuration object, `cfgL2CAP`, for a Bluetooth LE L2CAP signaling command frame or data frame.

`cfgL2CAP = bleL2CAPFrameConfig(Name,Value)` specifies properties on page 2-20 using one or more name-value pairs. Enclose each property name in quotes. For example, `('CommandType','Command reject')` sets the type of signaling command frame to `'Command reject'`.

Properties

Note For more information about the Bluetooth LE L2CAP frame properties and their respective values, see Volume 3, Part A, Section 3 of the Bluetooth Core Specification [2].

`ChannelIdentifier` — Identifier for logical channel endpoint

`'0005'` (default) | 4-element character vector | string scalar denoting a 2-octet hexadecimal value

Identifier for a logical channel endpoint, specified as a 4-element character vector or a string scalar denoting a 2-octet hexadecimal value. This property specifies the local name representing a logical channel endpoint. Use this value to identify the command and data frames. The command frames use `'0005'` as the `'ChannelIdentifier'`. The L2CAP B-frames use fixed `'ChannelIdentifier'`, `'0004'`, for attribute protocol (ATT) and `'0006'` for security manager protocol (SMP).

Data Types: `char` | `string`

`CommandType` — Signaling command type

`'Credit Based Connection request'` (default) | `'Command reject'` | `'Disconnection request'` | `...`

Signaling command type, specified as a character vector or a string scalar. Specify this property as one of these values:

- `'Command reject'`
- `'Disconnection request'`
- `'Disconnection response'`
- `'Connection parameter update request'`
- `'Connection parameter update response'`
- `'Credit based connection request'`
- `'Credit based connection response'`
- `'Flow control credit'`

Dependencies

To enable this property, set the ChannelIdentifier property to `'0005'`.

Data Types: `char` | `string`

`SignalIdentifier` — Identifier for request-response frame exchange

`'01'` (default) | 2-element character vector | string scalar denoting 1-octet hexadecimal value

Identifier for a request-response frame exchange, specified as a 2-element character vector or string scalar denoting a 1-octet hexadecimal value. The requesting device sets the value of this property and the responding device uses the same value in its response. The value of this property cannot be set to `'00'`.

Data Types: `char` | `string`

`CommandRejectReason` — Reason for rejecting received signaling command frame

`'Command not understood'` (default) | `'Signaling MTU exceeded'` | `'Invalid CID in request'`

Reason for rejecting the received signaling command frame, specified as a character vector or a string scalar. Specify this property as one of these values:

- `'Command not understood'`
- `'Signaling MTU exceeded'`
- `'Invalid CID in request'`

This property specifies the reason for rejecting a signaling command frame.

Data Types: `char` | `string`

`SourceChannelIdentifier` — Source logical channel endpoint

`'0040'` (default) | 4-element character vector | string scalar denoting 2-octet hexadecimal value

Source logical channel endpoint, specified as a 4-element character vector or string scalar denoting a 2-octet hexadecimal value. This property specifies the source channel endpoint from which the object send or receives request. When the channel is created using the credit-based connection procedure, the object sends the data packets flowing to the sender of the request to this value.

Data Types: `char` | `string`

`DestinationChannelIdentifier` — Destination logical channel endpoint
`'0040'` (default) | 4-element character vector | string scalar denoting 2-octet hexadecimal value

Destination logical channel endpoint, specified as a 4-element character vector or string scalar denoting a 2-octet hexadecimal value. This property specifies the destination channel endpoint from which the object send or receives request. When the channel is created using the credit-based connection procedure, the object sends the data packets flowing to the destination of the request to this value.

Data Types: `char` | `string`

`ConnectionIntervalRange` — Connection interval range
`[6,3200]` (default) | 2-element numeric vector specified as [*MIN*, *MAX*]

Connection interval range, specified as a 2-element numeric vector in the form of [*MIN*, *MAX*]. *MIN* and *MAX* specify the minimum and the maximum value of this property, respectively. Specify *MIN* and *MAX* values in the range [6, 3200]. *MIN* must be less than or equal to *MAX*. Each unit for *MIN* or *MAX* is taken as 1.25 ms so the resultant value is in the range [7.5 ms, 4.0 s].

Data Types: `double`

`ConnectionTimeout` — Connection supervision timeout
`10 (100 ms)` (default) | integer in the range [*Mct*, 3200]

Connection supervision timeout, specified as an integer in the range [*Mct*, 3200], where *Mct* is the larger of 10 and ((1+`PeripheralLatency`)×(`ConnectionInterval`×1.25)×2)/10. This property indicates the timeout for a connection if no valid packet is received within this time. Each unit is taken as 10 ms so that the resultant connection timeout, (`ConnectionInterval`×10), is in the range [100 ms, 32.0s].

Data Types: `double`

`PeripheralLatency` — Number of link layer connection events a slave can ignore
`0` (default) | integer in the range [0, *Msl*]

Number of link layer connection events a slave can ignore, specified as an integer in the range [0, *Msl*], where *Msl* is the lesser of 499 and ((`ConnectionTimeout`×10)/((`ConnectionInterval`×1.25)×2))-1.

Data Types: `double`

`ParameterUpdateResult` — Result of connection parameters update
`'Accepted'` (default) | `'Rejected'`

Result of the connection parameters update, specified as `'Accepted'` or `'Rejected'`. This property indicates the response to the `'Connection Parameter Update Request'` value of the `CommandType` property and specifies the result after updating the connection parameters.

Data Types: `char` | `string`

`LEPSM` — LE protocol or service multiplexer
`'001F'` (default) | 4-element character vector | string scalar denoting two-octet hexadecimal value

LE protocol or service multiplexer, specified as a 4-element character vector or a string scalar denoting a 2-octet hexadecimal value. The value of this property is a unique number specified by the Special Interest Group (SIG) for each protocol. The SIG assigns the value of this property within the

range [0x0001, 0x007F] for a set of existing protocols. The SIG dynamically assigns the value of this property in the range [0x0080, 0x00FF] to the implemented protocols.

Data Types: `char` | `string`

`MaxTransmissionUnit` — Maximum service data unit (SDU) size
23 (default) | integer in the range of [23, 65,535]

Maximum service data unit (SDU) size, specified as an integer in the range of [23, 65,535] octets. This property specifies the maximum acceptable SDU size for the upper-layer entity.

Data Types: `double`

`MaxPDUPayloadSize` — Maximum protocol data unit (PDU) payload size
23 (default) | integer in the range of [23, 65,535]

Maximum protocol data unit (PDU) payload size, specified as an integer in the range of [23, 65,535] octets. This property specifies the maximum acceptable payload data for the L2CAP layer entity.

Data Types: `double`

`Credits` — Number of LE frames peer device can send or receive
1 (default) | integer in the range of [0, 65,535]

Number of LE-frames peer device can send or receive, specified as an integer in the range of [0, 65,535] octets. This property indicates the number of LE-frames that the peer device can send or receive. If the value of the CommandType property is set to `'Flow control credit'`, then this property cannot be set to `0`.

Data Types: `double`

`ConnectionResult` — Result of credit-based connection procedure
`'Successful'` (default) | `'LEPSM not supported'` | `'No resources available'` | `...`

Result of the credit-based connection procedure, specified as a character vector or a string scalar. Specify this property as one of these values:

- `'Successful'`
- `'LEPSM not supported'`
- `'No resources available'`
- `'Insufficient authentication'`
- `'Insufficient authorization'`
- `'Insufficient encryption key size'`
- `'Insufficient encryption'`
- `'Invalid Source CID'`
- `'Source CID already allocated'`
- `'Unacceptable parameters'`

This property indicates the outcome of the connection request.

Data Types: `char` | `string`

Note From R2022a, this object uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Examples

Create Bluetooth LE L2CAP Frame Configuration Object Using Default Settings

Create a default Bluetooth LE L2CAP frame configuration object.

```
cfgL2CAP = bleL2CAPFrameConfig
```

```
cfgL2CAP =
  bleL2CAPFrameConfig with properties:

          ChannelIdentifier: '0005'
                CommandType: 'Credit based connection request'
           SignalIdentifier: '01'
    SourceChannelIdentifier: '0040'
                      LEPSM: '001F'
        MaxTransmissionUnit: 23
          MaxPDUPayloadSize: 23
                    Credits: 1
```

Set the value of credits to `10`.

```
cfgL2CAP.Credits = 10
```

```
cfgL2CAP =
  bleL2CAPFrameConfig with properties:

          ChannelIdentifier: '0005'
                CommandType: 'Credit based connection request'
           SignalIdentifier: '01'
    SourceChannelIdentifier: '0040'
                      LEPSM: '001F'
        MaxTransmissionUnit: 23
          MaxPDUPayloadSize: 23
                    Credits: 10
```

Create Bluetooth LE L2CAP Configuration Object Using Name-Value Arguments

Create a Bluetooth LE L2CAP frame configuration object, `'cfgL2CAP'`, by setting the value of channel identifier as `'0004'`using name-value pairs. This configuration object sets properties to generate a Bluetooth LE L2CAP data frame (B-frame).

```
cfgL2CAP = bleL2CAPFrameConfig('ChannelIdentifier','0004')
```

```
cfgL2CAP =
  bleL2CAPFrameConfig with properties:

    ChannelIdentifier: '0004'
```

End-to-End Workflow of Bluetooth LE L2CAP Frames

Create a Bluetooth LE L2CAP configuration object to generate a L2CAP data frame (B-frame). Set the value of channel identifier as `'0004'`.

```
cfgL2CAPTx = bleL2CAPFrameConfig('ChannelIdentifier','0004')
```

```
cfgL2CAPTx =
  bleL2CAPFrameConfig with properties:

    ChannelIdentifier: '0004'
```

Generate a Bluetooth LE L2CAP data frame (B-frame), specifying the service data unit (SDU) from ATT as `'0A0100'`.

```
l2capFrame = bleL2CAPFrame(cfgL2CAPTx,"0A0100")
```

```
l2capFrame = 7x2 char array
    '03'
    '00'
    '04'
    '00'
    '0A'
    '01'
    '00'
```

Decode the generated Bluetooth LE L2CAP data frame (B-frame). The returned status indicates decoding was successful.

```
[status,cfgL2CAPRx,SDU] = bleL2CAPFrameDecode(l2capFrame)
```

```
status =
  blePacketDecodeStatus enumeration

    Success


cfgL2CAPRx =
  bleL2CAPFrameConfig with properties:

    ChannelIdentifier: '0004'


SDU = 3x2 char array
    '0A'
    '01'
    '00'
```

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleL2CAPFrame` | `bleL2CAPFrameDecode`

Topics

“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”
“Bluetooth LE L2CAP Frame Generation and Decoding”

bleLLAdvertisingChannelPDUConfig

Bluetooth LE LL advertising channel PDU configuration parameters

Description

The `bleLLAdvertisingChannelPDUConfig` object parameterizes the `bleLLAdvertisingChannelPDU` function to generate a Bluetooth low energy (LE) link layer (LL) advertising channel protocol data unit (PDU).

Creation

Syntax

```
cfgLLAdv = bleLLAdvertisingChannelPDUConfig
cfgLLAdv = bleLLAdvertisingChannelPDUConfig(Name,Value)
```

Description

`cfgLLAdv = bleLLAdvertisingChannelPDUConfig` creates a default Bluetooth LE LL advertising channel PDU configuration object, `cfgLLAdv`.

`cfgLLAdv = bleLLAdvertisingChannelPDUConfig(Name,Value)` sets properties on page 2-27 using one or more name-value pairs. Enclose each property name in quotation marks. For example, `('PDUType','Scan response')` sets the type of Bluetooth LE LL advertising channel PDU to `'Scan response'`.

Properties

Note For more information about Bluetooth LE LL advertising channel PDU properties and their respective values, see Volume 6, Part B, Section 2.3 of the Bluetooth Core Specification [2].

PDUType — Bluetooth LE LL advertising channel PDU type

`'Advertising indication'` (default) | `'Advertising direct indication'` | `'Advertising non connectable indication'` | `...`

Bluetooth LE LL advertising channel PDU type, specified as a character vector or a string scalar. Specify this property as one of these values:

- `'Advertising indication'`
- `'Advertising direct indication'`
- `'Advertising non connectable indication'`
- `'Scan request'`
- `'Scan response'`

- `'Connection indication'`
- `'Advertising scannable indication'`

Data Types: `char` | `string`

`ChannelSelection` — Channel selection algorithm
`'Algorithm1'` (default) | `'Algorithm2'`

Channel selection algorithm, specified as `'Algorithm1'` or `'Algorithm2'`. This value indicates the type of algorithm that the object uses to hop between channels.

Data Types: `char` | `string`

`AdvertiserAddressType` — Advertiser device address type
`'Random'` (default) | `'Public'`

Advertiser device address type, specified as `'Random'` or `'Public'`.

Data Types: `char` | `string`

`AdvertiserAddress` — Advertiser device address
`'0123456789AB'` (default) | 12-element character vector | string scalar denoting a 6-octet hexadecimal value

Advertiser device address, specified as a 12-element character vector or a string scalar denoting a 6-octet hexadecimal value.

Data Types: `char` | `string`

`TargetAddressType` — Target device address type
`'Random'` (default) | `'Public'`

Target device address type, specified as `'Random'` or `'Public'`. This value indicates the type of target device address when a directed advertisement packet is transmitted.

Data Types: `char` | `string`

`TargetAddress` — Target device address
`'0123456789CD'` (default) | 12-element character vector | string scalar denoting a 6-octet hexadecimal value

Target device address, specified as a 12-element character vector or a string scalar denoting a 6-octet hexadecimal value. This value indicates the target device address when a directed advertisement packet is transmitted.

Data Types: `char` | `string`

`ScannerAddressType` — Scanner device address type
`'Random'` (default) | `'Public'`

Scanner device address type, specified as `'Random'` or `'Public'`. This value indicates the type of scanner device address when a scan request packet is transmitted.

Data Types: `char` | `string`

`ScannerAddress` — Scanner device address
`'0123456789CD'` (default) | 12-element character vector | string scalar denoting a 6-octet hexadecimal value

Scanner device address, specified as a 12-element character vector or a string scalar denoting a 6-octet hexadecimal value. This value indicates the scanner device address when a scan request packet is transmitted.

Data Types: `char` | `string`

`InitiatorAddressType` — Initiator device address type
`'Random'` (default) | `'Public'`

Initiator device address type, specified as `'Random'` or `'Public'`. This value indicates the type of initiator device address when a connection indication packet is transmitted.

Data Types: `char` | `string`

`InitiatorAddress` — Initiator device address
`'0123456789CD'` (default) | 12-element character vector | string scalar denoting a 6-octet hexadecimal value

Initiator device address, specified as a 12-element character vector or a string scalar denoting a 6-octet hexadecimal value. This value indicates the initiator device address when a connection indication packet is transmitted.

Data Types: `char` | `string`

`AdvertisingData` — Advertising data
`'020106'` (default) | character vector | string scalar | numeric vector of elements in the range [0,31] | *n*-by-2 character array

Advertising data, specified as one of these values:

- Character vector — This vector represent octets in hexadecimal format.
- String scalar — This scalar represent octets in hexadecimal format.
- Numeric vector of elements in the range [0,31] — This vector represent octets in decimal format.
- *n*-by-2 character array — Each row represent an octet in hexadecimal format.

This value indicates the advertising data that the device sends out in an advertisement packet.

Data Types: `char` | `string` | `double`

`ScanResponseData` — Scan response data
`'020106'` (default) | character vector | string scalar | numeric vector of elements in the range [0,31] | *n*-by-2 character array

Scan response data, specified as one of these values:

- Character vector — This vector represent octets in hexadecimal format.
- String scalar — This scalar represent octets in hexadecimal format.
- Numeric vector of elements in the range [0,31] — This vector represent octets in decimal format.
- *n*-by-2 character array — Each row represent an octet in hexadecimal format.

This value indicates the scan response data that the device sends out in a scan response packet (when scan request is received).

Data Types: `char` | `string` | `double`

`AccessAddress` — Unique connection address
`'01234567'` (default) | 8-element character vector or a string scalar

Unique connection address, specified as an 8-element character vector or a string scalar. This property indicates a unique 32-bit address that the LL generates for a new connection or periodic advertisement between two devices.

Data Types: `char` | `string`

`CRCInitialization` — Cyclic redundancy check (CRC) initialization value
`'012345'` (default) | 6-element character vector | 3-octet hexadecimal value

CRC initialization value, specified as a 6-element character vector or a string scalar denoting a 3-octet hexadecimal value. The object uses this property to initialize the CRC calculation.

Data Types: `char` | `string`

`WindowSize` — Transmit window size
`1` (default) | nonnegative integer

Transmit window size, specified as a nonnegative integer in the range [1, *Mws*], where *Mws* is the lesser of 8 and ConnectionInterval-1. This property indicates the window size within which the Central transmits the data packet and Peripheral listens for the data packet after connection establishment. Each unit is taken as 1.25 ms so that the (WindowSize×1.25) is in the range [1.25, min(10, ((`ConnectionInterval`×1.25) - 1.25) ms)] ms.

Data Types: `double`

`WindowOffset` — Transmit window offset
`0` (default) | nonnegative integer

Transmit window offset, specified as a nonnegative integer in the range [0, *Mwo*], where *Mwo* is the lesser of 3200 and `ConnectionInterval`. This value indicates the window offset after which the transmit window starts. Each unit is taken as 1.25 ms so that the resultant window offset (WindowOffset×1.25) is in the range of [0, (`ConnectionInterval`×1.25)] ms.

Data Types: `double`

`ConnectionInterval` — Connection interval
`6 (7.5 ms)` (default) | integer in the range [6, 3200]

Connection interval, specified as an integer in the range [6, 3200]. This value indicates the interval between the start of two consecutive connection events. Each unit is taken as 1.25 ms so that the resultant connection interval (`ConnectionInterval`×1.25) is in the range [7.5 ms, 4.0 s].

Data Types: `double`

`PeripheralLatency` — Peripheral latency
`0` (default) | nonnegative integer in the range [0, *Msl*]

Peripheral latency, specified as a nonnegative integer in the range [0, *Msl*], where *Msl* is the lesser of 499 and ((`ConnectionInterval`×10)/((`ConnectionInterval`×1.25)×2))-1. This value indicates the number of connection events that a Peripheral can skip listening for packets from the Central.

Data Types: `double`

`ConnectionTimeout` — Connection supervision timeout
`10` (default) | positive integer in the range [*Mct*, 3200]

Connection supervision timeout, specified as a positive integer in the range [*Mct*, 3200], where *Mct* is the greater of 10 and ((1+`PeripheralLatency`)×(`ConnectionInterval`×1.25)×2)/10. This value indicates the timeout for a connection if no valid packet is received within this time. Each unit is taken as 10 ms so that the resultant connection timeout (`ConnectionInterval`×10) is in the range of [100 ms, 32.0 s].

Data Types: `double`

`UsedChannels` — List of used data channels
row vector containing the channel indices between [0:36] (default) | integer vector with element values in the range [0, 36]

List of used data channels, specified as an integer vector with element values in the range [0, 36]. The vector length must be greater than 1. At least two channels must be set as used (good) channels. This value indicates the set of good channels that the Central classifies.

Data Types: `double`

`HopIncrement` — Hop increment count
5 (default) | integer in the range [5, 16]

Hop increment count, specified as an integer in the range [5, 16]. This property indicates the hop increment count that the object uses to hop between data channels.

Data Types: `double`

`SleepClockAccuracy` — Master sleep clock accuracy
`'251 to 500 ppm'` (default) | `'151 to 250 ppm'` | `'101 to 150 ppm'` | `...`

Master sleep clock accuracy, specified as a character vector or a string scalar indicating the worst case Central sleep clock accuracy. Specify this property as one of these values:

- `'251 to 500 ppm'`
- `'151 to 250 ppm'`
- `'101 to 150 ppm'`
- `'76 to 100 ppm'`
- `'51 to 75 ppm'`
- `'31 to 50 ppm'`
- `'21 to 30 ppm'`
- `'0 to 20 ppm'`

Data Types: `char` | `string`

Note From R2022a, this object uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Examples

Create Bluetooth LE LL Advertising Channel PDU Configuration Object

Create a default Bluetooth LE LL advertising channel configuration object.

```
cfgLLAdv = bleLLAdvertisingChannelPDUConfig

cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                  PDUType: 'Advertising indication'
         ChannelSelection: 'Algorithm1'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '0123456789AB'
          AdvertisingData: [3x2 char]
```

Create Bluetooth LE LL Advertising Channel PDU Configuration Object Using Name-Value Arguments

Create two unique Bluetooth LE LL advertising channel configuration objects of type `'Scan response'` and `'Connection indication'` using name-value arguments.

Create a Bluetooth LE LL advertising channel PDU configuration object by setting the values of PDU type to `'Scan response'`, advertiser address to `'1234567890AB'`, and scan response data to `'020106020AD3'`.

```
cfgLLAdv = bleLLAdvertisingChannelPDUConfig('PDUType','Scan response', ...
    'AdvertiserAddress','1234567890AB', ...
    'ScanResponseData','020106020AD3')

cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                  PDUType: 'Scan response'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '1234567890AB'
         ScanResponseData: [6x2 char]
```

Create another Bluetooth LE LL advertising channel PDU configuration object and specify the type of PDU as `'Connection indication'`. Set the values of the connection interval as `64` and the set of data channels as `[0 4 12 16 18 24 25]`.

```
cfgLLAdv = bleLLAdvertisingChannelPDUConfig('PDUType','Connection indication');
cfgLLAdv.ConnectionInterval = 64;
cfgLLAdv.UsedChannels = [0 4 12 16 18 24 25]

cfgLLAdv =
  bleLLAdvertisingChannelPDUConfig with properties:

                  PDUType: 'Connection indication'
         ChannelSelection: 'Algorithm1'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '0123456789AB'
     InitiatorAddressType: 'Random'
         InitiatorAddress: '0123456789CD'
            AccessAddress: '01234567'
       CRCInitialization: '012345'
              WindowSize: 1
```

```
            WindowOffset: 0
        ConnectionInterval: 64
         PeripheralLatency: 0
         ConnectionTimeout: 10
              UsedChannels: [0 4 12 16 18 24 25]
              HopIncrement: 5
        SleepClockAccuracy: '251 to 500 ppm'
```

End-to-End Workflow of Bluetooth LE LL Advertising Channel PDU

Create a Bluetooth LE LL advertising channel PDU configuration object, specifying the values of PDU type as `'Connection indication'`, the connection interval as `8`, and the set of data channels as `[0 4 12 16 18 24 25]`.

```
cfgLLAdvTx = bleLLAdvertisingChannelPDUConfig('PDUType','Connection indication');
cfgLLAdvTx.ConnectionInterval = 8;
cfgLLAdvTx.UsedChannels = [0 4 12 16 18 24 25]
```

```
cfgLLAdvTx =
  bleLLAdvertisingChannelPDUConfig with properties:

                  PDUType: 'Connection indication'
         ChannelSelection: 'Algorithm1'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '0123456789AB'
     InitiatorAddressType: 'Random'
         InitiatorAddress: '0123456789CD'
            AccessAddress: '01234567'
        CRCInitialization: '012345'
               WindowSize: 1
            WindowOffset: 0
        ConnectionInterval: 8
         PeripheralLatency: 0
         ConnectionTimeout: 10
              UsedChannels: [0 4 12 16 18 24 25]
              HopIncrement: 5
        SleepClockAccuracy: '251 to 500 ppm'
```

Generate a Bluetooth LE LL advertising channel PDU by using the corresponding configuration object.

```
pdu = bleLLAdvertisingChannelPDU(cfgLLAdvTx);
```

Decode the generated Bluetooth LE LL advertising channel PDU. The returned status indicates decoding is successful.

```
[status,cfgLLAdvRx] = bleLLAdvertisingChannelPDUDecode(pdu)
```

```
status =
  blePacketDecodeStatus enumeration

    Success
```

```
cfgLLAdvRx =
  bleLLAdvertisingChannelPDUConfig with properties:

                 PDUType: 'Connection indication'
       ChannelSelection: 'Algorithm1'
    AdvertiserAddressType: 'Random'
        AdvertiserAddress: '0123456789AB'
     InitiatorAddressType: 'Random'
         InitiatorAddress: '0123456789CD'
            AccessAddress: '01234567'
       CRCInitialization: '012345'
              WindowSize: 1
            WindowOffset: 0
      ConnectionInterval: 8
       PeripheralLatency: 0
       ConnectionTimeout: 10
            UsedChannels: [0 4 12 16 18 24 25]
            HopIncrement: 5
      SleepClockAccuracy: '251 to 500 ppm'
```

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleLLAdvertisingChannelPDU` | `bleLLAdvertisingChannelPDUDecode`

Topics

“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”
“Bluetooth LE Link Layer Packet Generation and Decoding”

bleLLControlPDUConfig

Bluetooth LE LL control PDU payload configuration parameters

Description

The `bleLLControlPDUConfig` object parameterizes the `bleLLDataChannelPDU` function to generate a Bluetooth low energy (LE) link layer (LL) control protocol data unit (PDU) payload.

Creation

Syntax

```
cfgControl = bleLLControlPDUConfig
cfgControl = bleLLControlPDUConfig(Name,Value)
```

Description

`cfgControl = bleLLControlPDUConfig` creates a default Bluetooth LE LL control PDU payload configuration object.

`cfgControl = bleLLControlPDUConfig(Name,Value)` sets properties on page 2-35 by using one or more name-value pairs. Enclose each property name in quotes. For example, `('Opcode','Version indication')` sets the operation code as `'Version indication'`.

Properties

Note For more information about Bluetooth LE LL control PDU properties and their respective values, see Volume 6, Part B, Section 2.4 of the Bluetooth Core Specification [2].

Opcode — Bluetooth LE LL control PDU payload configuration operation code

`'Connection update indication'` (default) | `'Channel map indication'` | `'Terminate indication'` | `'Unknown response'` | `'Version indication'` | `'Reject indication'`

Bluetooth LE LL control PDU payload configuration operation code, specified as one of these values.

- `'Connection update indication'`
- `'Channel map indication'`
- `'Terminate indication'`
- `'Unknown response'`
- `'Version indication'`
- `'Reject indication'`

Data Types: `char` | `string`

`WindowSize` — Transmit window size
`1` (default) | nonnegative integer in the range [1, *Mws*]

Transmit window size, specified as a nonnegative integer in the range [1, *Mws*], where *Mws* is the lesser of 8 and (ConnectionInterval-1). This property indicates the window size within which the Central transmits a data packet and the Peripheral listens for a data packet after the connection is established. Each unit is taken as 1.25 ms so that the window size (WindowSize×1.25) is in the range [1.25, min(10, ((`ConnectionInterval`×1.25) - 1.25))] ms.

Data Types: `double`

`ConnectionInterval` — Connection interval
`6 (7.5 ms)` (default) | integer in the range [6, 3200]

Connection interval, specified as an integer in the range [6, 3200]. This property indicates the interval between the start of two consecutive connection events. Each unit is taken as 1.25 ms so that the connection interval (`ConnectionInterval`×1.25) is in the range of [7.5 ms, 4.0 s].

Data Types: `double`

`PeripheralLatency` — Peripheral latency
`0` (default) | nonnegative integer in the range [0, *Msl*]

Peripheral latency, specified as a nonnegative integer in the range [0, *Msl*], where *Msl* is the lesser of 499 and ((`ConnectionTimeout`×10)/((`ConnectionInterval`×1.25)×2))-1. This property indicates the number of connection events that a Peripheral can ignore.

Data Types: `double`

`ConnectionTimeout` — Connection supervision timeout
`10` (default) | nonnegative integer in the range [*Mct*, 3200]

Connection supervision timeout, specified as a nonnegative integer in the range [*Mct*, 3200], where *Mct* is the larger of 10 and ((1+PeripheralLatency)*(`ConnectionInterval`×1.25)×2)/10. If the Peripheral does not receive a valid packet within this time, this property indicates the connection timeout. Each unit is taken as 10 ms so that the connection timeout (`ConnectionInterval`×10) is in the range [100 ms, 32.0 s].

Data Types: `double`

`Instant` — Connection event instant
`0` (default) | integer in the range [0, 65535]

Connection event instant, specified as an integer in the range [0, 65535]. This property indicates the event count at which specific action must occur.

Data Types: `double`

`UsedChannels` — List of used data channels
row vector containing the channel indices between [0:36] (default) | integer vector with element values in the range [0, 36]

List of used data channels, specified as an integer vector with element values in the range [0, 36]. The vector length must be greater than 1. At least two channels must be set as used (good) channels. This property indicates the set of good channels that the Central classifies.

Data Types: `double`

ErrorCode — Connection termination error code
`'Success'` (default) | `'Unknown connection identifier'` | `'Hardware failure'` | ...

Connection termination error code, specified as one of these values. Each valid value describes the error description informing the remote device why the connection is about to be terminated.

- `'Success'`
- `'Unknown connection identifier'`
- `'Hardware failure'`
- `'Memory capacity exceeded'`
- `'Connection timeout'`
- `'Connection limit exceeded'`
- `'Connection already exists'`
- `'Command disallowed'`
- `'Connection accept timeout exceeded'`
- `'Connection rejected due to limited resources'`
- `'Invalid LL parameters'`
- `'Connection rejected due to unacceptable BD_ADDR'`
- `'Unspecified error'`
- `'Unsupported LL parameter value'`
- `'Role change not allowed'`
- `'LL response timeout'`
- `'LL procedure collision'`
- `'Instant passed'`
- `'Channel classification not supported'`
- `'Extended inquiry response too large'`
- `'Connection rejected due to no suitable channel found'`
- `'Advertising timeout'`
- `'Controller busy'`
- `'Unacceptable connection parameters'`
- `'Connection failed to be established'`
- `'Unknown advertising identifier'`
- `'Limit reached'`
- `'Operation cancelled by host'`

Data Types: `char` | `string`

UnknownOpcode — Unrecognized or unsupported operation code
`'00'` (default) | 2-element character vector | string scalar denoting 1-octet hexadecimal value

Unrecognized or unsupported operation code, specified as a 2-element character vector or string scalar denoting a 1-octet hexadecimal value. This property indicates the type of Bluetooth LE LL control PDU that the object does not recognize or support.

Data Types: `char` | `string`

`VersionNumber` — Version number of Bluetooth Core Specification
`'5.0'` (default) | `'4.0'` | `'4.1'` | `'4.2'`

Version number of Bluetooth Core Specification, specified as `'4.0'`, `'4.1'`, `'4.2'`, or `'5.0'`. This property indicates the version number of the Bluetooth Core Specification.

Data Types: `string`

`CompanyIdentifier` — Manufacturer ID of Bluetooth controller
`'FFFF'` (default) | 4-element character vector | string scalar denoting a 2-octet hexadecimal value

Manufacturer ID of Bluetooth controller, specified as a 4-element character vector or a string scalar denoting a 2-octet hexadecimal value. This property indicates the unique identifier assigned to your organization by Bluetooth Special Interest Group (SIG) [2].

Data Types: `char` | `string`

`SubVersionNumber` — Subversion number of the Bluetooth controller
`'0000'` (default) | 4-element character vector | string scalar denoting a 2-octet hexadecimal value

Subversion number of the Bluetooth controller, specified as a 4-element character vector or string scalar denoting a 2-octet hexadecimal value. This property indicates the unique value for each implementation or revision of a Bluetooth controller implementation.

Data Types: `char` | `string`

Note From R2022a, this object uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Examples

Create Bluetooth LE LL Control PDU Configuration Object

Create a default Bluetooth LE LL control PDU configuration object.

```
cfgControl = bleLLControlPDUConfig
```

```
cfgControl =
  bleLLControlPDUConfig with properties:

                Opcode: 'Connection update indication'
            WindowSize: 1
          WindowOffset: 0
    ConnectionInterval: 6
     PeripheralLatency: 0
     ConnectionTimeout: 10
               Instant: 0
```

Set the value of the connection interval to 64.

```
cfgControl.ConnectionInterval = 64
```

```
cfgControl =
  bleLLControlPDUConfig with properties:
```

```
                Opcode: 'Connection update indication'
            WindowSize: 1
          WindowOffset: 0
    ConnectionInterval: 64
     PeripheralLatency: 0
     ConnectionTimeout: 10
               Instant: 0
```

Create Bluetooth LE LL Control PDU Configuration Object Using Name-Value Arguments

Create two unique Bluetooth LE LL control PDU configuration objects using name-value arguments: first one of the type `'Terminate indication'` and error code `'Connection timeout'` and second one of the type `'Channel map indication'` with used set of data channels.

Create a Bluetooth LE LL control PDU configuration object, specifying opcode as `'Terminate indication'` and error code as `'Connection timeout'`.

```
cfgControl = bleLLControlPDUConfig('Opcode','Terminate indication', ...
    'ErrorCode','Connection Timeout')
```

```
cfgControl =
  bleLLControlPDUConfig with properties:

       Opcode: 'Terminate indication'
    ErrorCode: 'Connection timeout'
```

Create another Bluetooth LE LL control PDU payload configuration object, this time by setting the value of opcode as `'Channel map indication'`. Specify the list of used data channels.

```
cfgControl = bleLLControlPDUConfig('Opcode','Channel map indication');
cfgControl.UsedChannels = [0 3 12 16 18 24]
```

```
cfgControl =
  bleLLControlPDUConfig with properties:

          Opcode: 'Channel map indication'
         Instant: 0
    UsedChannels: [0 3 12 16 18 24]
```

Create a Bluetooth LE LL data channel configuration object, specifying the values of `'LLID'` as `'Control'` and `'ControlConfig'` as `'cfgControl'`.

```
cfgLLData = bleLLDataChannelPDUConfig('LLID','Control');
cfgLLData.ControlConfig = cfgControl
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Control'
                NESN: 0
      SequenceNumber: 0
```

```
            MoreData: 0
    CRCInitialization: '012345'
        ControlConfig: [1x1 bleLLControlPDUConfig]
```

End-to-End Workflow of Bluetooth LE LL Control PDU

Create a default Bluetooth LE LL control PDU configuration object.

```
cfgControl = bleLLControlPDUConfig
```

```
cfgControl =
  bleLLControlPDUConfig with properties:

               Opcode: 'Connection update indication'
           WindowSize: 1
         WindowOffset: 0
    ConnectionInterval: 6
     PeripheralLatency: 0
     ConnectionTimeout: 10
               Instant: 0
```

Create a Bluetooth LE LL data channel PDU configuration object, specifying the link layer identifier, `'LLID'`, as `'Control'` and `'ControlConfig'` as `'cfgControl'`. Specify the cyclic redundancy check (CRC).

```
cfgLLData = bleLLDataChannelPDUConfig('LLID','Control', ...
    'ControlConfig',cfgControl);
cfgLLData.CRCInitialization = 'E23456'
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Control'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
    CRCInitialization: 'E23456'
        ControlConfig: [1x1 bleLLControlPDUConfig]
```

Generate a Bluetooth LE LL control PDU.

```
pdu = bleLLDataChannelPDU(cfgLLData);
```

Decode the generated Bluetooth LE LL control PDU by initializing the CRC value. The returned status indicates decoding is successful.

```
crcInit = 'E23456';                                % Received during associaton
[status,cfgRx,llPayload] = bleLLDataChannelPDUDecode(pdu,crcInit)
```

```
status =
  blePacketDecodeStatus enumeration
```

```
    Success


cfgRx =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Control'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
    CRCInitialization: '012345'
        ControlConfig: [1x1 bleLLControlPDUConfig]


llPayload =

  1x0 empty char array
```

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2019. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleLLDataChannelPDU` | `bleLLDataChannelPDUDecode`

Objects

`bleLLDataChannelPDUConfig`

Topics

“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”
“Bluetooth LE Link Layer Packet Generation and Decoding”

bleLLDataChannelPDUConfig

Bluetooth LE LL data channel PDU configuration parameters

Description

The `bleLLDataChannelPDUConfig` parameterizes the `bleLLDataChannelPDU` function to generate a Bluetooth low energy (LE) link layer (LL) data channel protocol data unit (PDU).

Creation

Syntax

```
cfgLLData = bleLLDataChannelPDUConfig
cfgLLData = bleLLDataChannelPDUConfig(Name,Value)
```

Description

`cfgLLData = bleLLDataChannelPDUConfig` creates a default Bluetooth LE LL data channel PDU configuration object, `cfgLLData`.

`cfgLLData = bleLLDataChannelPDUConfig(Name,Value)` sets properties on page 2-42 by using one or more name-value pairs. Enclose each property name in quotes. For example, `('LLID','Data (start fragment/complete)')` sets the type of Bluetooth LE LL data channel PDU as a data PDU.

Properties

Note For more information about Bluetooth LE LL data channel PDU properties and their respective values, see Volume 6, Part B, Section 2.4 of the Bluetooth Core Specification [2].

`LLID` — Link layer identifier

`'Data (continuation fragment/empty)'` (default) | `'Data (start fragment/complete)'` | `'Control'`

Link layer identifier, specified as a character vector or a string scalar to describe the type of a Bluetooth LE LL data channel PDU. Specify this property as one of these values:

- `'Data (continuation fragment/empty)'`
- `'Data (start fragment/complete)'`
- `'Control'`

Data Types: `char` | `string`

`NESN` — Next expected sequence number

`0` or `false` | `1` or `true`

Next expected sequence number, specified as `1` (`true`) or `0` (`false`). This property indicates either an acknowledgment for the last received packet or a request for resending the last transmitted packet from the peer.

Data Types: `logical`

`SequenceNumber` — Transmitting packet sequence number
`0` or `false` | `1` or `true`

Transmitting packet sequence number, specified as `1` (`true`) or `0` (`false`). This property indicates the sequence number of the transmitted packet.

Data Types: `logical`

`MoreData` — Indication for more data
`0` or `false` | `1` or `true`

Indication for more data, specified as `1` (`true`) or `0` (`false`). A `1` (`true`) value indicates that the sender has more data to send.

Data Types: `logical`

`CRCInitialization` — Cyclic redundancy check (CRC) initialization value
`'012345'` (default) | 6-element character vector | string scalar denoting 3-octet hexadecimal value

CRC initialization value, specified as a 6-element character vector or a string scalar denoting a 3-octet hexadecimal value.

Data Types: `char` | `string`

`ControlConfig` — Bluetooth LE LL control PDU payload configuration object
`bleLLControlPDUConfig` object

Bluetooth LE LL control PDU payload configuration object, specified as a `bleLLControlPDUConfig` object. The object configures the fields of this value by using the properties of `bleLLControlPDUConfig` object. The object uses this property to generate and decode a Bluetooth LE LL control PDU.

Examples

Create Bluetooth LE LL Data Channel PDU Configuration Object for Data PDU

Create a default Bluetooth LE LL data channel PDU configuration object for a data PDU.

```
cfgLLData = bleLLDataChannelPDUConfig
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (continuation fragment/empty)'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
    CRCInitialization: '012345'
```

Set the value of LLID to `'Data (start fragment/complete)'`.

```
cfgLLData.LLID = 'Data (start fragment/complete)'
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (start fragment/complete)'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
   CRCInitialization: '012345'
```

Create Bluetooth LE LL Data Channel PDU Configuration Object for Control PDU

Create two unique Bluetooth LE LL data channel PDU configuration objects for a control PDU of type `'Channel map indication'` and `'Version indication'`.

Create a Bluetooth LE LL control PDU payload configuration object for a control PDU with opcode `'Channel map indication'`.

```
cfgControl = bleLLControlPDUConfig('Opcode','Channel map indication')
```

```
cfgControl =
  bleLLControlPDUConfig with properties:

         Opcode: 'Channel map indication'
        Instant: 0
   UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ... ]
```

Create a Bluetooth LE LL data channel configuration object by specifying the values of `'LLID'` as `'Control'` and `'ControlConfig'` as `'cfgControl'`.

```
 cfgLLData = bleLLDataChannelPDUConfig('LLID','Control', ...
    'ControlConfig',cfgControl)
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Control'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
   CRCInitialization: '012345'
       ControlConfig: [1x1 bleLLControlPDUConfig]
```

Create another Bluetooth LE LL data channel configuration object for a control PDU, this time specifying the opcode as `'Version indication'`.

```
cfgControl.Opcode = 'Version indication';
cfgControl.SubVersionNumber = '008E'
```

```
cfgControl =
  bleLLControlPDUConfig with properties:
```

```
              Opcode: 'Version indication'
       VersionNumber: '5.0'
    CompanyIdentifier: 'FFFF'
     SubVersionNumber: '008E'
```

Create a Bluetooth LE LL data channel configuration object, specifying the values of `'ControlConfig'` as `'cfgControl'`.

```
cfgLLData.ControlConfig = cfgControl
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Control'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
    CRCInitialization: '012345'
       ControlConfig: [1x1 bleLLControlPDUConfig]
```

End-to-End Workflow of Bluetooth LE LL Data Channel PDU

Create a default Bluetooth LE LL data channel PDU configuration object for a data PDU.

```
cfgLLData = bleLLDataChannelPDUConfig
```

```
cfgLLData =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (continuation fragment/empty)'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
    CRCInitialization: '012345'
```

Initialize the CRC value.

```
cfgLLData.CRCInitialization = '123456';
```

Generate a Bluetooth LE LL data channel PDU by using the upper-layer payload, `'030004000A0100'`.

```
pdu = bleLLDataChannelPDU(cfgLLData,'030004000A0100');
```

Decode the generated Bluetooth LE LL data channel PDU by initializing the CRC value. The returned status indicates decoding is successful.

```
crcInit = '123456';
[status,cfgRx,llPayload] = bleLLDataChannelPDUDecode(pdu,crcInit)
```

```
status =
  blePacketDecodeStatus enumeration
```

```
    Success


cfgRx =
  bleLLDataChannelPDUConfig with properties:

                LLID: 'Data (continuation fragment/empty)'
                NESN: 0
      SequenceNumber: 0
            MoreData: 0
    CRCInitialization: '012345'


llPayload = 7x2 char array
    '03'
    '00'
    '04'
    '00'
    '0A'
    '01'
    '00'
```

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleLLDataChannelPDU` | `bleLLDataChannelPDUDecode`

Objects

`bleLLControlPDUConfig`

Topics

“Bluetooth Packet Structure”
“Generate and Decode Bluetooth Protocol Data Units”

“Bluetooth LE Link Layer Packet Generation and Decoding”

blePCAPWriter

PCAP or PCAPNG file writer of Bluetooth LE LL packets

Description

The `blePCAPWriter` object writes generated and recovered Bluetooth low energy (LE) link layer (LL) packets to a packet capture (PCAP) or packet capture next generation (PCAPNG) file (`.pcap` or `.pcapng`, respectively).

Creation

Syntax

```
obj = blePCAPWriter
obj = blePCAPWriter(Name,Value)
```

Description

`obj = blePCAPWriter` creates a default Bluetooth LE PCAP or PCAPNG file writer object that writes Bluetooth LE LL packets to a PCAP or PCAPNG file, respectively.

`obj = blePCAPWriter(Name,Value)` sets properties on page 2-48 using one or more name-value pairs. Enclose each property name in quotes. For example, `('FileExtension','pcapng')` sets the extension of the file as `.pcapng`.

Properties

Note The `blePCAPWriter` object does not overwrite the existing PCAP or PCAPNG file. Each time when you create this object, specify a unique PCAP or PCAPNG file name.

`FileName` — Name of the PCAP or PCAPNG file
`'bleCapture'` (default) | character row vector | string scalar

Name of the PCAP or PCAPNG file, specified as a character row vector or a string scalar.

Data Types: `char` | `string`

`ByteOrder` — Byte order
`'little-endian'` (default) | `'big-endian'`

Byte order, specified as `'little-endian'` or `'big-endian'`.

Data Types: `char` | `string`

`FileExtension` — Type of file
`'pcap'` (default) | `'pcapng'`

Type of file, specified as `'pcap'` or `'pcapng'`.

Data Types: `char` | `string`

`FileComment` — Comment for PCAPNG file
`''` (default) | character vector | string scalar

Comment for the PCAPNG file, specified as a character vector or a string scalar.

Data Types: `char` | `string`

`Interface` — Name of interface on which Bluetooth LE packets are captured
`'BLE'` (default) | character vector | string scalar

Name of the interface on which Bluetooth LE packets are captured, specified as a character vector or a string scalar.

Data Types: `char` | `string`

`PhyHeaderPresent` — Flag to indicate presence of PHY header
`false` or `0` (default) | `true` or `1`

Flag to indicate the presence of physical layer (PHY) header, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`PCAPWriter` — PCAP or PCAPNG file writer object
`pcapWriter` object | `pcapngWriter` object

PCAP or PCAPNG file writer object, specified as `pcapWriter` or `pcapngWriter` object.

Object Functions

Specific to This Object

write Write Bluetooth LE LL protocol packet data to PCAP or PCAPNG file

Examples

Use Bluetooth LE PCAP Writer to Write LL Packet to PCAP File

Create a Bluetooth LE PCAP file writer object, specifying the name of the PCAP file.

```
pcapObj = blePCAPWriter('FileName','writeblepacket');
```

Generate a Bluetooth LE LL packet.

```
cfgLLData = bleLLDataChannelPDUConfig('LLID', ...
    'Data (start fragment/complete)');
payload = '0E00050014010A001F004000170017000000';
llDataPDU = bleLLDataChannelPDU(cfgLLData,payload);
connAccessAddress = de2bi(hex2dec('E213BC42'),32)';
llpacket = [connAccessAddress; llDataPDU];
```

Write the Bluetooth LE LL packet to the PCAP file.

```
timestamp = 0;                                          % Packet arrival time in POSIX® microseco
write(pcapObj,llpacket,timestamp,'PacketFormat','bits');
```

Use Bluetooth LE PCAP Writer to Write LL Packet to PCAPNG File

Create a Bluetooth LE PCAPNG file writer object, specifying the name and extension of the PCAPNG file.

```
pcapObj = blePCAPWriter('FileName','sampleBLELL', ...
    'FileExtension','pcapng');
```

Generate a Bluetooth LE LL packet.

```
cfgLLData = bleLLDataChannelPDUConfig('LLID', ...
    'Data (start fragment/complete)');
payload = '0E00050014010A001F004000170017000000';
llDataPDU = bleLLDataChannelPDU(cfgLLData,payload);
connAccessAddress = de2bi(hex2dec('E213BC42'),32)';
llpacket = [connAccessAddress; llDataPDU];
```

Write the Bluetooth LE LL packet to the PCAPNG file.

```
timestamp = 12800000;                                % Packet arrival time in POSIX® microseconds
write(pcapObj,llpacket,timestamp,'PacketFormat','bits');
```

Write Bluetooth LE LL Packet to PCAPNG File from PCAPNG File Writer

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapObj = pcapngWriter('FileName','sampleBLELL', ...
    'FileComment','This is a sample file');
```

Create a Bluetooth LE PCAP file writer object, specifying the PCAPNG file writer and the presence of PHY header.

```
blePCAP = blePCAPWriter('PCAPWriter',pcapObj,'PhyHeaderPresent',true);
```

Generate a Bluetooth LE LL packet.

```
cfgLLAdv = bleLLAdvertisingChannelPDUConfig;
cfgLLAdv.PDUType = 'Advertising indication';
cfgLLAdv.AdvertisingData = '020106';
llDataPDU = bleLLAdvertisingChannelPDU(cfgLLAdv);
connAccessAddress = de2bi(hex2dec('E213BC42'),32)';
llpacket = [connAccessAddress;llDataPDU];
```

Write the Bluetooth LE LL packet to the PCAPNG file.

```
PhyHeaderBytes = [39 10 8 1 10 10 10 10 15 00];
timestamp = 18912345;                              % Packet arrival time in POSIX® microseconds ela
write(blePCAP,llpacket,timestamp,'PacketFormat','bits', ...
    'PhyHeader',PhyHeaderBytes, ...
    'PacketComment','This is the first packet');
```

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`pcapWriter` | `pcapngWriter`

Topics

“Bluetooth LE Link Layer Packet Generation and Decoding”
“Bluetooth LE L2CAP Frame Generation and Decoding”

bluetoothConnectionConfig

Bluetooth BR connection configuration parameters

Description

Use the `bluetoothConnectionConfig` object to set the baseband connection configuration parameters for a Bluetooth basic rate (BR) Central and Peripheral node pair.

Creation

Syntax

```
cfgConnection = bluetoothConnectionConfig
cfgConnection = bluetoothConnectionConfig(Name=Value)
```

Description

`cfgConnection = bluetoothConnectionConfig` creates a default Bluetooth BR connection configuration object that shares the baseband connection configuration parameters between a Central and a Peripheral node.

`cfgConnection = bluetoothConnectionConfig(Name=Value)` sets properties on page 2-52 by using one or more name-value arguments. For example, `HoppingSequenceType="Connection adaptive"` sets the frequency hopping sequence type to `"Connection adaptive"`.

Properties

`CentralToPeripheralACLPacketType` — Packet type used over ACL logical transport from Central to Peripheral
`"DH1"` (default) | `"DM1"` | `"DM3"` | `"DH3"` | `"DM5"` | `"DH5"`

Packet type used over the asynchronous connection-oriented logical (ACL) logical transport from Central to Peripheral node, specified as `"DH1"`, `"DM1"`, `"DM3"`, `"DH3"`, `"DM5"`, or `"DH5"`.

Data Types: `char` | `string`

`PeripheralToCentralACLPacketType` — Packet type used over ACL logical transport from Peripheral to Central
`"DH1"` (default) | `"DM1"` | `"DM3"` | `"DH3"` | `"DM5"` | `"DH5"`

Packet type used over ACL logical transport from Peripheral to Central node, specified as `"DH1"`, `"DM1"`, `"DM3"`, `"DH3"`, `"DM5"`, or `"DH5"`.

Data Types: `char` | `string`

`SCOPacketType` — Packet type transmitted and received over SCO logical transport
`"None"` (default) | `"HV1"` | `"HV2"` | `"HV3"`

Packet type transmitted and received over the synchronous connection-oriented (SCO) logical transport, specified as `"None"`, `"HV1"`, `"HV2"`, or `"HV3"`. If you set this property to `"None"`, the object disables the SCO link, and no voice transmission occurs between the Bluetooth BR nodes. If you set this property to `"HV1"`, `"HV2"`, or `"HV3"`, the object enables the synchronous link. Based on the configured SCO packet type, the Bluetooth BR node internally generates voice traffic with a data rate of 64 Kbps. To send `"DV"` packets over the SCO link, perform these steps.

1. Set this property to `"HV1"`.
2. Attach an ACL traffic source for the data payload with a packet size in the range of [1, 9] bytes.

The object supports one SCO link in a physical link and up to three SCO links at the Central node.

Data Types: `char` | `string`

`HoppingSequenceType` — Frequency hopping sequence type
`"Connection basic"` (default) | `"Connection adaptive"`

Frequency hopping sequence type, specified as `"Connection basic"` or `"Connection adaptive"`.

Data Types: `char` | `string`

`UsedChannels` — List of used (good) data channels
`[0:78]` (default) | vector of integers in range [0, 78]

List of used (good) data channels for the Bluetooth BR physical link to use, specified as a vector of integers in the range [0, 78]. This value specifies the indices of the assigned data channels. To ensure that at least 20 channels are set as used (good) channels, specify this vector with unique values and a length greater than or equal to 20.

Dependencies

To enable this property, set the HoppingSequenceType property to `"Connection adaptive"`.

Data Types: `double`

`PollInterval` — Maximum interval between two successive poll requests
`40` (default) | integer in range [6, 4096]

Maximum interval between two successive poll requests, specified as an integer in the range [6, 4096]. Set this value in number of slots. For more information about this property, see the Bluetooth Core Specification [2], Volume 2, Part C, Section 5.4.

Data Types: `double`

`InstantOffset` — Slots after which new channel map is enforced
`240` (default) | integer in range [96, 69120000]

Slots after which the new channel map is enforced, specified as an integer in the range [96, 69120000]. The object adds this value to the current clock and updates the channel map. Units are in slots. This value must be at least `6×PollInterval` slots or 96 slots, whichever is greater.

Dependencies

To enable this property, set the HoppingSequenceType property to `"Connection adaptive"`.

Data Types: `double`

`TransmitterPower` — Packet transmission power
`0` (default) | scalar in range [–20, 20]

Packet transmission power, specified as a scalar in the range [–20, 20]. The transmitter applies this value on the packet before sending it to the antenna. Units are in dBm.

Data Types: `double`

`SupervisionTimeout` — Connection supervision timeout
`32000` (default) | scalar in range [400, 65535] | `Inf`

Connection supervision timeout, specified as a scalar in the range [400, 65535] or `Inf`. Units are in slots.

- If the Bluetooth BR node does not receive a valid packet within the time set by this value, the object times out the connection. After the timeout, the logical link is disconnected at the baseband layer and no connection exists between the nodes.
- If you set this value to Inf, the connection is always active.

Data Types: `double`

`CentralAddress` — Bluetooth BR Central node address
`blanks(0)` (default) | 12-element character vector | string scalar denoting 6-octet hexadecimal value

This property is read-only.

Bluetooth BR Central node address, stored as a 12-element character vector or a string scalar denoting a 6-octet hexadecimal value. This property sets a unique address for the Bluetooth BR Central node.

Data Types: `char` | `string`

`PrimaryLTAddress` — Primary logical transport address
`[]` (default) | positive integer scalar

This property is read-only.

Primary logical transport address, stored as a positive integer scalar.

Data Types: `double`

Object Functions

Specific to This Object

configureConnection Configure connection between Bluetooth BR Central and Peripheral nodes

Examples

Create, Configure, and Simulate Bluetooth BR Nodes

Initialize the wireless network simulator.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and the other with the `"peripheral"` role. Specify the position of the Peripheral node, in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between the Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure the connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode)
```

```
connection =
  bluetoothConnectionConfig with properties:

    CentralToPeripheralACLPacketType: "DH1"
    PeripheralToCentralACLPacketType: "DH1"
                         SCOPacketType: "None"
                  HoppingSequenceType: "Connection adaptive"
                         UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 ... ]
                         PollInterval: 40
                        InstantOffset: 240
                     TransmitterPower: 20
                   SupervisionTimeout: 32000

   Read-only properties:
                       CentralAddress: "D091BBE70001"
                     PrimaryLTAddress: 1
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
    GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Specify the simulation time, in seconds.

```
simulationTime = 0.3;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve the application, baseband, and physical layer (PHY) statistics corresponding to the Central and Peripheral nodes.

```
centralStats = statistics(centralNode)

centralStats = struct with fields:
        Name: "Node1"
          ID: 1
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]


peripheralStats = statistics(peripheralNode)

peripheralStats = struct with fields:
        Name: "Node2"
          ID: 2
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

Version History

Introduced in R2022b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed May 22, 2022. https://www.bluetooth.com/.

[2] Bluetooth Core Specifications Working Group. "Bluetooth Core Specification" v5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothNode`

Topics
“Create, Configure, and Simulate Bluetooth BR Piconet”
“Simulate Multiple Bluetooth BR Piconets with ACL Traffic”
“Bluetooth Node Statistics”

bluetoothPathLossConfig

Bluetooth BR/EDR or LE path loss configuration parameters

Description

The `bluetoothPathLossConfig` object specifies parameters for the `bluetoothPathLoss` function. Use the `bluetoothPathLoss` function to estimate the path loss between Bluetooth basic rate/enhanced data rate (BR/EDR) or low energy (LE) devices.

Creation

Syntax

```
cfgPathLoss = bluetoothPathLossConfig
cfgPathLoss = bluetoothPathLossConfig(Name=Value)
```

Description

`cfgPathLoss = bluetoothPathLossConfig` creates a default Bluetooth BR/EDR or LE path loss estimation configuration object.

`cfgPathLoss = bluetoothPathLossConfig(Name=Value)` sets properties on page 2-57 by using one or more name-value arguments. For example, `Environment="Home"` sets the signal propagation environment to home.

Properties

`Environment` — Signal propagation environment

`"Outdoor"` (default) | `"Home"` | `"Industrial"` | `"Office"`

Signal propagation environment, specified as `"Home"`, `"Industrial"`, `"Office"`, or `"Outdoor"`.

Data Types: `char` | `string`

`TransmitterAntennaGain` — Transmitter antenna gain

`0` (default) | scalar or a row vector of values in the range [–10, 10]

Transmitter antenna gain, specified as a scalar or a row vector of values in the range [–10, 10]. Units are in dBi.

Data Types: `double`

`ReceiverAntennaGain` — Receiver antenna gain

`0` (default) | scalar or a row vector of values in the range [–10, 10]

Receiver antenna gain, specified as a scalar or a row vector of values in the range [–10, 10]. Units are in dBi.

Data Types: `double`

`TransmitterCableLoss` — Transmitter cable loss
`0` (default) | nonnegative scalar or a row vector

Transmitter cable loss, specified as a nonnegative scalar or a row vector. Units are in dB.

Data Types: `double`

`ReceiverCableLoss` — Receiver cable loss
`0` (default) | nonnegative scalar or a row vector

Receiver cable loss, specified as a nonnegative scalar or a row vector. Units are in dB.

Data Types: `double`

`TransmitterAntennaHeight` — Transmitter antenna height
`1` (default) | positive scalar or a row vector

Transmitter antenna height, specified as a positive scalar or a row vector. Units are in meters.

Dependencies

To enable this property, set the Environment property to `"Outdoor"`.

Data Types: `double`

`ReceiverAntennaHeight` — Receiver antenna height
`1` (default) | positive scalar or a row vector

Receiver antenna height, specified as a positive scalar or a row vector. Units are in meters.

Dependencies

To enable this property, set the Environment property to `"Outdoor"`.

Data Types: `double`

`PathLossExponent` — Path loss exponent of the specified environment
`2.2` (default) | positive scalar

Path loss exponent of the specified environment, specified as a positive scalar.

Dependencies

To enable this property, set the Environment property to `"Industrial"`.

Data Types: `double`

`StandardDeviation` — Standard deviation
`2.667` (default) | nonnegative scalar

Standard deviation, specified as a nonnegative scalar. Units are in dB.

Dependencies

To enable this property, set the Environment property to `"Industrial"` or `"Outdoor"`.

Data Types: `double`

`RandomStream` — Random number source
`"Global stream"` (default) | `"mt19937ar with seed"`

Random number source, specified as `"Global stream"` or `"mt19937ar with seed"`.

- `"Global stream"` — The object generates a normally distributed random number by using the current global random number stream.
- `"mt19937ar with seed"` — The object generates a normally distributed random number by using the mt19937ar algorithm.

Data Types: `char` | `string`

Seed — Initial seed for mt19937ar
73 (default) | nonnegative scalar

Initial seed for mt19937ar, specified as a nonnegative scalar.

Dependencies

To enable this property, set the RandomStream property to `"mt19937ar with seed"`.

Data Types: `double`

`PathLossModel` — Path loss model of the specified environment
`'TwoRayGroundReflection'` | `'LogNormalShadowing'` | `'NISTPAP02Task6'`

This property is read-only.

Path loss model of the specified environment, returned as `'TwoRayGroundReflection'`, `'LogNormalShadowing'`, or `'NISTPAP02Task6'`.

- If you set the Environment property to `"Outdoor"`, this property returns the `'TwoRayGroundReflection'` value.
- If you set the `Environment` property to `"Industrial"`, this property returns the `'LogNormalShadowing'` value.
- If you set the `Environment` property to `"Home"` or `"Office"`, this property returns the `'NISTPAP02Task6'` value.

Data Types: `char`

Examples

Create Bluetooth Path Loss Estimation Configuration Objects

Create a default Bluetooth path loss estimation configuration object. By default, the object specifies an outdoor environment.

```
cfgPathLossOutdoor = bluetoothPathLossConfig;
```

Specify the transmitter and receiver antenna heights, in meters.

```
cfgPathLossOutdoor.TransmitterAntennaHeight = 2;
cfgPathLossOutdoor.ReceiverAntennaHeight = 3;
```

Specify the standard deviation, in dB.

```
cfgPathLossOutdoor.StandardDeviation = 1.6667
```

```
cfgPathLossOutdoor =
  bluetoothPathLossConfig with properties:

                Environment: 'Outdoor'
      TransmitterAntennaGain: 0
         ReceiverAntennaGain: 0
        TransmitterCableLoss: 0
           ReceiverCableLoss: 0
    TransmitterAntennaHeight: 2
       ReceiverAntennaHeight: 3
           StandardDeviation: 1.6667
                RandomStream: 'Global stream'

   Read-only properties:
               PathLossModel: 'TwoRayGroundReflection'
```

Create a Bluetooth path loss estimation configuration object for an industrial environment, specifying the path loss exponent as `2` and standard deviation as `0` dB. A path loss exponent value of `2` and a standard deviation of `0` dB signifies that the Bluetooth signal propagates in free space.

```
cfgPathLossIndustrial = bluetoothPathLossConfig(Environment="Industrial", ...
    PathLossExponent=2,StandardDeviation=0)
```

```
cfgPathLossIndustrial =
  bluetoothPathLossConfig with properties:

               Environment: 'Industrial'
    TransmitterAntennaGain: 0
       ReceiverAntennaGain: 0
      TransmitterCableLoss: 0
         ReceiverCableLoss: 0
          PathLossExponent: 2
         StandardDeviation: 0
              RandomStream: 'Global stream'

   Read-only properties:
             PathLossModel: 'LogNormalShadowing'
```

Version History

Introduced in R2022b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed March 22, 2022. https://www.bluetooth.com/.

[2] Bluetooth Core Specification 5.3. Bluetooth Special Interest Group (SIG), Accessed March 22, 2022. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bluetoothPathLoss`

bluetoothFrequencyHop

Bluetooth BR/EDR channel index for frequency hopping

Description

The `bluetoothFrequencyHop` object creates a Bluetooth basic rate/enhanced data rate (BR/EDR) channel index for frequency hopping. Use this object to generate the hopping sequence for inquiry, paging, and connection procedures.

Creation

Syntax

```
freqHop = bluetoothFrequencyHop
freqHop = bluetoothFrequencyHop(Name,Value)
```

Description

`freqHop = bluetoothFrequencyHop` creates a default Bluetooth BR/EDR channel index object for frequency hopping.

`freqHop = bluetoothFrequencyHop(Name,Value)` sets “Properties” on page 2-62 by using one or more name-value pairs. Enclose each property name in quotes. For example, `('SequenceType','Page')` sets the frequency hopping sequence type to `Page`.

Properties

`DeviceAddress` — Bluetooth BR/EDR device address

`'9E8B33'` (default) | 12-element character vector | string scalar denoting a 6-octet hexadecimal value

Bluetooth BR/EDR device address, specified as a 12-element character vector or a string scalar denoting a 6-octet hexadecimal value. This value specifies the Bluetooth BR/EDR device address given as an input to the hop selection kernel. This property ignores all the consecutive 0s starting from the most significant bit (MSB). This table maps the value of this property to different physical channels.

Type of Physical Channel	Value of `DeviceAddress` Property
Basic	Address of Central
Page scan	Address of the scanning device
Inquiry	General inquiry access code (GIAC)

The default value of this property denotes the lower address part (LAP) of GIAC.

Data Types: `char` | `string`

SequenceType — Frequency hopping sequence type

`'Inquiry'` (default) | `'Page scan'` | `'Inquiry scan'` | `'Page'` | `...`

Frequency hopping sequence type, specified as one of these values:

- `'Page scan'`
- `'Inquiry scan'`
- `'Page'`
- `'Inquiry'`
- `'Central page response'`
- `'Peripheral page response'`
- `'Inquiry response'`
- `'Connection basic'`
- `'Connection adaptive'`
- `'Interlaced page scan'`
- `'Interlaced inquiry scan'`

Data Types: `char` | `string`

InterlaceOffset — Offset for available frequencies in inquiry and paging procedures

16 (default) | integer in the range [0, 31]

Offset for available frequencies in inquiry and paging procedures, specified as an integer in the range [0, 31].

Dependencies

To enable this property, set the "SequenceType" on page 2-0 property to `'Interlaced page scan'` or `'Interlaced inquiry scan'`.

Data Types: `double`

KNudge — Offset to compute control signal (X)

`0` (default) | even integer

Offset to compute control signal (X), specified as an even integer. This property specifies the additional offset that the object adds to the clock bits.

Dependencies

To enable this property, set the "SequenceType" on page 2-0 property to `'Page'` or `'Inquiry'`.

Data Types: `double`

KOffset — Offset to switch between A-train and B-train

24 (default) | 8

Offset to switch between A-train and B-train, specified as 24 (for A-train) or 8 (for B-train). To switch between the trains, this property specifies the offset added to the clock bits.

Dependencies

To enable this property, set the "SequenceType" on page 2-0 property to `'Page'` or `'Inquiry'`.

Data Types: `double`

`Counter` — Counter for Central or Peripheral page response sequence
`0` (default) | nonnegative integer

Counter for Central or Peripheral page response sequence, specified as a nonnegative integer. This property is incremented at every Central transmission slot.

Dependencies

To enable this property, set the “SequenceType” on page 2-0 property to `'Peripheral page response'`, `'Central page response'`, or `'Inquiry response'`.

Data Types: `double`

`UsedChannels` — List of used channels
row vector containing all 79 channel indices (default) | vector of integers in the range [0, 78]

List of used channels, specified as a vector of integers in the range [0, 78]. The vector must contain at least 20 elements.

Dependencies

To enable this property, set the “SequenceType” on page 2-0 property to `'Connection adaptive'`.

Data Types: `double`

Note From R2022a, this object uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Object Functions

Specific to This Object

nextHop Select Bluetooth BR/EDR channel index to hop for next frequency

Examples

Select Bluetooth BR/EDR Channel Index for Connection Basic Frequency Hopping Sequence

Create a Bluetooth BR/EDR channel index object for frequency hopping.

```
freqHop = bluetoothFrequencyHop;
```

Specify the frequency hopping sequence type as connection basic.

```
freqHop.SequenceType = 'Connection basic';
```

Specify a clock value and Bluetooth BR/EDR device address.

```
inputClock = '2C';                % 28-bit
freqHop.DeviceAddress = '2A96EF25'
```

```
freqHop =
  bluetoothFrequencyHop with properties:

    DeviceAddress: '2A96EF25'
     SequenceType: 'Connection basic'
```

Select a Bluetooth BR/EDR channel index to hop for next frequency.

```
[channelIndex, X] = nextHop(freqHop,inputClock)
```

```
channelIndex = 27
```

```
X = 11
```

Select Bluetooth BR/EDR Channel Index for Connection Adaptive Frequency Hopping Sequence

Create a Bluetooth BR/EDR channel index object for frequency hopping, specifying frequency hopping sequence type, Bluetooth BR/EDR device address, and used channels.

```
freqHop = bluetoothFrequencyHop('SequenceType','Connection adaptive', ...
    'DeviceAddress','2A96EF25','UsedChannels',22:78)
```

```
freqHop =
  bluetoothFrequencyHop with properties:

    DeviceAddress: '2A96EF25'
     SequenceType: 'Connection adaptive'
     UsedChannels: [22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 ... ]
```

Specify a clock value.

```
inputClock = '12C';      % 28-bit
```

Select a Bluetooth BR/EDR channel index to hop for next frequency.

```
[channelIndex,X] = nextHop(freqHop,inputClock)
```

```
channelIndex = 65
```

```
X = 11
```

Select Bluetooth BR/EDR Channel Index for Page Frequency Hopping Sequence

Create a Bluetooth BR/EDR channel index object for frequency hopping.

```
freqHop = bluetoothFrequencyHop;
```

Specify the frequency hopping sequence type as page.

```
freqHop.SequenceType = 'Page';
```

Specify a clock value, offset to select frequencies in A-train, and Bluetooth BR/EDR device address.

```
inputClock = 44;               % 28-bit
freqHop.KOffset = 24;
freqHop.DeviceAddress = '2A96EF25'
```

```
freqHop =
  bluetoothFrequencyHop with properties:

    DeviceAddress: '2A96EF25'
     SequenceType: 'Page'
           KNudge: 0
          KOffset: 24
```

Select a Bluetooth BR/EDR channel index to hop for next frequency.

```
[channelIndex,X] = nextHop(freqHop,inputClock)
```

```
channelIndex = 15

X = 30
```

Select Bluetooth BR/EDR Channel Index for Peripheral Page Response Frequency Hopping Sequence

Create a default Bluetooth BR/EDR channel index object for frequency hopping.

```
freqHop = bluetoothFrequencyHop;
```

Specify the frequency hopping sequence type as `'Peripheral page response'`.

```
freqHop.SequenceType = 'Peripheral page response';
```

Specify a clock value, counter for Peripheral page response, and Bluetooth BR/EDR device address.

```
frozenClock = '2A';                % 28-bit
freqHop.Counter = 1;
freqHop.DeviceAddress = '2A96EF25'
```

```
freqHop =
  bluetoothFrequencyHop with properties:

    DeviceAddress: '2A96EF25'
     SequenceType: 'Peripheral page response'
          Counter: 1
```

Select a Bluetooth BR/EDR channel index to hop for the next frequency.

```
[channelIndex,X] = nextHop(freqHop,frozenClock)
```

```
channelIndex = 28

X = 1
```

Version History

Introduced in R2020b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`bleChannelSelection`

bluetoothLEBIGConfig

Bluetooth LE BIG configuration parameters

Description

Use `bluetoothLEBIGConfig` object to set the broadcast isochronous group (BIG) configuration parameters between an isochronous broadcaster and a synchronized receiver.

Creation

Syntax

```
cfgBIG = bluetoothLEBIGConfig
cfgBIG = bluetoothLEBIGConfig(Name=Value)
```

Description

`cfgBIG = bluetoothLEBIGConfig` creates a default Bluetooth low energy (LE) BIG configuration object.

`cfgBIG = bluetoothLEBIGConfig(Name=Value)` sets properties on page 2-68 by using one or more name-value arguments. For example, `ISOInterval=0.01` sets the isochronous event interval to 0.01 seconds.

Properties

`SeedAccessAddress` — Seed access address of the BIG
`"78E52493"` (default) | 8-element character vector | string scalar denoting a 4-octet hexadecimal value

Seed access address of the BIG, specified as an 8-element character vector or a string scalar denoting a 4-octet hexadecimal value. This property specifies the seed access address from which the link layer of Bluetooth LE node derives the broadcast isochronous streams (BIS) access address. For more information about seed access address, see Volume 6, Part B, Section 2.1.2 of Bluetooth Core Specification v5.3 [2].

Data Types: `char` | `string`

`PHYMode` — Physical layer (PHY) mode for transmission or reception
`"LE1M"` (default) | `"LE2M"` | `"LE125K"` | `"LE500K"`

PHY mode for transmission or reception, specified as `"LE1M"`, `"LE2M"`, `"LE125K"`, or `"LE500K"`.

Data Types: `char` | `string`

`NumBIS` — Number of BISes in BIG
`1` (default) | integer in the range [1, 31]

Number of BISes in BIG, specified as an integer in the range [1, 31]. For more information about the number of BIS, see Volume 6, Part B, Section 4.4.6.3 of Bluetooth Core Specification v5.3 [2].

Data Types: `double`

`ISOInterval` — Isochronous event interval
`0.005` (default) | scalar in the range [0.005, 4]

Isochronous event interval, specified as a scalar in the range of [0.005, 4]. Specify this value in seconds. Set this value as a multiple of 1.25 milliseconds. This property specifies the time between two adjacent BIG anchor points. For more information about isochronous event interval, see Volume 6, Part B, Section 4.4.6.3 of Bluetooth Core Specification v5.3 [2].

Data Types: `double`

`BISSpacing` — Time interval between successive BIS events
`0.002238` (default) | scalar in the range [0.000198, 1.048575]

Time interval between successive BIS events, specified as a scalar in the range of [0.000198, 1.048575]. Specify this value in seconds. This property specifies the time between the start of corresponding subevents in the adjacent BIS events present in the BIG and the start of the first subevent of the last BIS and the control subevent (if present). For more information about the BIS spacing, see Volume 6, Part B, Section 4.4.6.3 of Bluetooth Core Specification v5.3 [2].

Data Types: `double`

`SubInterval` — Time interval between successive subevents
`0.002238` (default) | scalar in the range [0.000198, 1.048575]

Time interval between successive subevents, specified as a scalar in the range of [0.000198, 1.048575]. Specify this value in seconds. This property specifies the time between the start of two consecutive subevents of each BIS. For more information about sub interval, see Volume 6, Part B, Section 4.4.6.3 of Bluetooth Core Specification v5.3 [2].

Data Types: `double`

`MaxPDU` — Maximum payload length
`251` (default) | integer in the range [1, 251]

Maximum payload length, specified as an integer in the range of [1, 251]. This property specifies the maximum number of data octets that can be included in each BIS data PDU. The object does not include the message integrity check in the data octets. For more information about maximum payload length, see Volume 6, Part B, Section 4.4.6.3 of Bluetooth Core Specification v5.3 [2].

Data Types: `double`

`BurstNumber` — Number of payloads associated with BIS event
`1` (default) | integer in the range [1, 7]

Number of payloads associated with BIS event, specified as an integer in the range of [1, 7]. For more information about burst number, see Volume 6, Part B, Section 4.4.6.3 of Bluetooth Core Specification v5.3 [2].

Data Types: `double`

`PreTransmissionOffset` — Pretransmission offset
`0` (default) | integer in the range [0, 15]

Pretransmission offset, specified as an integer in the range of [0, 15]. The subevents carrying pretransmissions contain data associated with the future BIS events that this property specifies. For more information about pretransmission offset, see Volume 6, Part B, Section 4.4.6.3 of Bluetooth Core Specification v5.3 [2].

Data Types: `double`

`RepetitionCount` — Immediate repetition count
`1` (default) | integer in the range [1, 15]

Immediate repetition count, specified as an integer in the range of [1, 15]. This property specifies the number of times a payload is transmitted in a BIS event. For more information about immediate repetition count, see Volume 6, Part B, Section 4.4.6.3 of Bluetooth Core Specification v5.3 [2].

Data Types: `double`

`NumSubevents` — Number of subevents in each BIS event of BIG
`1` (default) | integer in the range [1, 31]

Number of subevents in each BIS event of the BIG, specified as an integer in the range of [1, 31]. For more information about the number of subevents, see Volume 6, Part B, Section 4.4.6.3 of Bluetooth Core Specification v5.3 [2].

Data Types: `double`

`BISArrangement` — Arrangement of BIS events in the BIG
`"sequential"` (default) | `"interleaved"`

Arrangement of BIS events in the BIG, specified as `"sequential"` or `'interleaved'`. In `"sequential"` BIS arrangement, all the subevents of a BIS event occur together sequentially as a group. In `"interleaved"` BIS arrangement, corresponding subevents of each BIS event occur adjacent to each other. For more information about the arrangement of BIS events, see Volume 6, Part B, Section 4.4.6.4 of Bluetooth Core Specification v5.3 [2].

Data Types: `char` | `string`

`BIGOffset` — Offset of the anchor point of the first BIG event
`0` (default) | finite nonnegative scalar

Offset of the anchor point of the first BIG event, specified as a finite nonnegative scalar. Specify this value in seconds. This property specifies the time after which the first BIG event starts. For more information about BIG offset, see Volume 6, Part B, Section 4.4.6.11 of Bluetooth Core Specification v5.3 [2].

Data Types: `double`

`ReceiveBISNumbers` — List of BIS indices to be received by receiver
`1` (default) | integer vector with element values in the range [1, 31]

List of BIS indices to be received by the receiver, specified as an integer vector with element values in the range [1, 31].

Data Types: `double`

`UsedChannels` — List of used (good) data channels
`[0:36]` (default) | integer vector with element values in the range [0, 36]

List of used (good) data channels, specified as an integer vector with element values in the range [0, 36]. This value specifies the indices of the assigned data channels. This property indicates the set of good channels used by the connection. To ensure that at least two channels are set as used (good) channels, specify a vector length greater than 1.

Data Types: `double`

`InstantOffset` — Offset added to the BIG event counter
6 (default) | integer in the range [6, 65535]

Offset added to the BIG event counter, specified as an integer in the range [6, 65535]. This property specifies the offset added to the BIG event count. The BIG control procedures, such as the channel map update, take effect after InstantOffset BIG events from the current BIG event. For more information about instant offset, see Volume 6, Part B, Section 2.6.3 of Bluetooth Core Specification v5.3 [2]

Data Types: `double`

`BaseCRCInitialization` — Base cyclic redundancy check (CRC) initialization
`"1234"` (default) | 4-element character vector | string scalar denoting a 2-octet hexadecimal value

Base CRC initialization, specified as a 4-element character vector or string scalar denoting a 2-octet hexadecimal value. This property specifies the most significant two octets of the CRC initialization vector. For more information about base CRC initialization, see Volume 6, Part B, Section 3.1.1 of Bluetooth Core Specification v5.3 [2].

Data Types: `char` | `string`

Object Functions

Specific to This Object

configureBIG Configure BIG parameters at Bluetooth LE isochronous broadcaster and receiver

Examples

Create and Configure Bluetooth LE Node with BIG Configuration Properties

Create a Bluetooth LE node, specifying the role as `"isochronous-broadcaster"`.

```
isoBroadcasterNode = bluetoothLENode("isochronous-broadcaster");
```

Create a Bluetooth LE node, specifying the role as `"synchronized-receiver"`.

```
receiverNode = bluetoothLENode("synchronized-receiver");
```

Create a default BIG configuration object.

```
bigConfig = bluetoothLEBIGConfig;
```

Specify the number of BISes in the BIG, the number of subevents in each BIS event in the BIG, and the BIS arrangement.

```
bigConfig.NumBIS = 2;
bigConfig.NumSubevents = 2;
bigConfig.BISArrangement = "interleaved";
```

Specify the number of payloads associated with a BIS event.

```
bigConfig.BurstNumber = 2;
```

Specify the time interval between successive BIS subevents and the isochronous event interval.

```
bigConfig.SubInterval = 0.006; % In seconds
bigConfig.ISOInterval = 0.015  % In seconds
```

```
bigConfig =
  bluetoothLEBIGConfig with properties:

        SeedAccessAddress: "78E52493"
                  PHYMode: "LE1M"
                   NumBIS: 2
              ISOInterval: 0.0150
               BISSpacing: 0.0022
              SubInterval: 0.0060
                   MaxPDU: 251
              BurstNumber: 2
    PretransmissionOffset: 0
          RepetitionCount: 1
             NumSubevents: 2
           BISArrangement: "interleaved"
                BIGOffset: 0
        ReceiveBISNumbers: 1
             UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ... ]
            InstantOffset: 6
    BaseCRCInitialization: "1234"
```

Assign the BIG configuration to the Bluetooth LE nodes.

```
configureBIG(bigConfig,isoBroadcasterNode,receiverNode);
```

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 12, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothLENode` | `bluetoothLEConnectionConfig` | `bluetoothMeshProfileConfig` | `bluetoothMeshFriendshipConfig`

Topics
“Bluetooth LE Audio”
“Create, Configure, and Simulate Bluetooth LE Network”
“Create, Configure, and Simulate Bluetooth LE Broadcast Audio Network”
“Create and Visualize Bluetooth LE Broadcast Audio Residential Scenario”
“Estimate Packet Delivery Ratio of LE Broadcast Audio in Residential Scenario”

bluetoothLEConnectionConfig

Bluetooth LE LL connection configuration parameters

Description

Use `bluetoothLEConnectionConfig` object to set the link layer (LL) connection configuration parameters at a Bluetooth low energy (LE) Central or Peripheral node.

Creation

Syntax

```
cfgConnection = bluetoothLEConnectionConfig
cfgConnection = bluetoothLEConnectionConfig(Name=Value)
```

Description

`cfgConnection = bluetoothLEConnectionConfig` creates a default Bluetooth LE connection configuration object that shares the LL connection configuration parameters between a Central node and a Peripheral node.

`cfgConnection = bluetoothLEConnectionConfig(Name=Value)` sets properties on page 2-74 by using one or more name-value arguments. For example, `ConnectionInterval=0.04` sets the connection interval to 0.04 seconds.

Properties

`ConnectionInterval` — LL connection interval

`0.02` (default) | scalar in the range [0.0075, 4]

LL connection interval, specified as a scalar in the range [0.0075, 4]. Specify this value in seconds. This property indicates the interval between the start of two consecutive LL connection events. Set this value as an integer multiple of 1.25 milliseconds. For more information about connection interval, see Volume 6, Part B, Section 4.5.1 of Bluetooth Core Specification v5.3 [2].

Data Types: `double`

`AccessAddress` — Unique connection address

`"5DA44270"` (default) | 8-element character vector | string scalar denoting a 4-octet hexadecimal value

Unique connection address, specified as an 8-element character vector or a string scalar denoting a 4-octet hexadecimal value. This property specifies a unique 32-bit access address for the LL connection between a Central node and a Peripheral node.

Data Types: `char` | `string`

`UsedChannels` — List of used (good) data channels

`[0:36]` (default) | integer vector with element values in the range [0, 36]

List of used data channels, specified as an integer vector with element values in the range [0, 36]. This value specifies the indices of the assigned data channels. This property indicates the set of good channels used by the connection. To ensure that at least two channels are set as used (good) channels, specify a vector length greater than 1.

Data Types: `double`

`Algorithm` — Channel selection algorithm
`1` (default) | `2`

Algorithm, specified as `1` or `2` representing "Algorithm #1" or "Algorithm #2", respectively. This property specifies the channel selection algorithm for LL connection events.

Data Types: `double`

`HopIncrement` — Hop increment count
`5` (default) | integer in the range [5, 16]

Hop increment count, specified as an integer in the range [5, 16]. This property specifies the number of hops between data channels. If you set the Algorithm property to 1, the object uses this value as an input.

Data Types: `double`

`CRCInitialization` — Cyclic redundancy check (CRC) initialization
`"012345"` (default) | 6-element character vector | string scalar denoting a 3-octet hexadecimal value

CRC initialization, specified as a 6-element character vector or a string scalar denoting a 3-octet hexadecimal value.

Data Types: `char` | `string`

`SupervisionTimeout` — LL connection supervision timeout
`1` (default) | scalar in the range [0.1, 32]

LL connection supervision timeout, specified as a scalar in the range [0.1, 32]. Specify this value in seconds. This property specifies the timeout for a LL connection if no valid packet is received within this time. Set this value as a multiple of 10 milliseconds.

Data Types: `double`

`PHYMode` — Physical layer (PHY) mode for generating or decoding
`"LE1M"` (default) | `"LE2M"` | `"LE125K"` | `"LE500K"`

PHY mode for generating or decoding, specified as `"LE1M"`, `"LE2M"`, `"LE125K"`, or `"LE500K"`.

Data Types: `char` | `string`

`InstantOffset` — Offset added to the connection event counter
`6` (default) | integer in the range [6, 65535]

Offset added to the connection event counter, specified as an integer in the range [6, 65535]. Specify this value in seconds. The object updates the channel map after `InstantOffset` connection events from the current connection event.

Data Types: `double`

`ConnectionOffset` — Offset for starting connection event
`0` (default) | integer in the range [0, `ConnectionInterval`]

Offset for starting connection event, specified as a scalar in the range [0, ConnectionInterval]. Specify this value in seconds. Within a connection interval, after the connection offset, communication begins between the Central and Peripheral nodes.

Data Types: `double`

`ActivePeriod` — Active communication period in each connection event
`0.020` (default) | integer in the range [0, `ConnectionInterval`]

Active communication period in each connection event, specified as a scalar in the range [0, ConnectionInterval]. Specify this value in seconds. Within a connection interval, this property specifies when communication begins between the Central and Peripheral nodes.

Data Types: `double`

Object Functions

Specific to This Object

configureConnection Configure LL connection between Bluetooth LE Central and Peripheral nodes

Examples

Create and Configure LL Connection Between Bluetooth LE Nodes

Create a Bluetooth LE node, specifying the role as `"central"`.

```
centralNode = bluetoothLENode("central");
```

Create two Bluetooth LE nodes, specifying the role as `"peripheral"`.

```
peripheralNode1 = bluetoothLENode("peripheral");
peripheralNode2 = bluetoothLENode("peripheral");
```

Create a default Bluetooth LE configuration object to share connection between the Central and Peripheral nodes.

```
connectionConfig = bluetoothLEConnectionConfig
```

```
connectionConfig =
  bluetoothLEConnectionConfig with properties:

    ConnectionInterval: 0.0200
         AccessAddress: "5DA44270"
          UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
             Algorithm: 1
          HopIncrement: 5
     CRCInitialization: "012345"
    SupervisionTimeout: 1
               PHYMode: "LE1M"
         InstantOffset: 6
      ConnectionOffset: 0
```

```
          ActivePeriod: 0.0200
```

Specify the connection interval, connection offset, and active period for the LL connection.

```
connectionConfig.ConnectionInterval = 0.04; % In seconds
connectionConfig.ConnectionOffset = 0;      % In seconds
connectionConfig.ActivePeriod = 0.02;       % In seconds
```

Specify the unique connection address for the LL connection.

```
connectionConfig.AccessAddress = "5DA44271"
```

```
connectionConfig =
  bluetoothLEConnectionConfig with properties:

    ConnectionInterval: 0.0400
         AccessAddress: "5DA44271"
          UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
             Algorithm: 1
          HopIncrement: 5
     CRCInitialization: "012345"
    SupervisionTimeout: 1
               PHYMode: "LE1M"
         InstantOffset: 6
      ConnectionOffset: 0
          ActivePeriod: 0.0200
```

Configure the LL connection between the Central node and Peripheral node 1.

```
configureConnection(connectionConfig,centralNode,peripheralNode1);
```

Update the LL connection configuration object to share connection parameters between the Central node and Peripheral node 2.

```
connectionConfig.ConnectionOffset = 0.02;   % In seconds
connectionConfig.ActivePeriod = 0.02;       % In seconds
connectionConfig.AccessAddress = "5DA44272"
```

```
connectionConfig =
  bluetoothLEConnectionConfig with properties:

    ConnectionInterval: 0.0400
         AccessAddress: "5DA44272"
          UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
             Algorithm: 1
          HopIncrement: 5
     CRCInitialization: "012345"
    SupervisionTimeout: 1
               PHYMode: "LE1M"
         InstantOffset: 6
      ConnectionOffset: 0.0200
          ActivePeriod: 0.0200
```

Configure the LL connection between the Central node and Peripheral node 2.

```
configureConnection(connectionConfig,centralNode,peripheralNode2);
```

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 16, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Objects

`bluetoothLENode` | `bluetoothLEBIGConfig` | `bluetoothMeshProfileConfig` | `bluetoothMeshFriendshipConfig` | `bleChannelSelection`

Topics

“Create, Configure, and Simulate Bluetooth LE Network”
“Create, Configure, and Simulate Bluetooth LE Broadcast Audio Network”
“Create, Configure and Simulate Bluetooth Mesh Network”
“Estimate Packet Delivery Ratio of LE Broadcast Audio in Residential Scenario”
“Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network”

bluetoothLENode

Bluetooth LE node

Description

Use the `bluetoothLENode` object to create and configure a Bluetooth low energy (LE) node.

Creation

Syntax

```
LENode = bluetoothLENode(Role)
LENode = bluetoothLENode(Role,Name=Value)
```

Description

`LENode = bluetoothLENode(Role)` creates a default Bluetooth LE node object for the specified role `Role`.

`LENode = bluetoothLENode(Role,Name=Value)` sets properties on page 2-79 of the Bluetooth LE node object by using one or more optional name-value arguments. For example, `"central",TransmitterPower=0` sets the transmitter power of the Central node to 0 dBm.

Properties

`Name` — Node name
`"Node`*N*`"` (default) | character vector | string scalar

Node name, specified as a character vector or string scalar. The default format of this value is `"Node`*N*`"`, where *N* is the node identifier specified by the ID property.

Data Types: `char` | `string`

`Position` — Position in 3-D Cartesian coordinates
`[0 0 0]` (default) | numeric row vector of length three

Position in 3-D Cartesian coordinates, specified as a numeric row vector of length three. Specify this value in meters. This value specifies the position of the node in Cartesian x-, y-, z-coordinates.

Data Types: `double`

`TransmitterPower` — Signal transmission power
`20` (default) | scalar in the range [-20, 20]

Signal transmission power, specified as a scalar in the range [-20, 20]. Specify this value in dBm. This value specifies the average power that the transmitter applies to the Bluetooth LE signal before sending it to the antenna.

Data Types: `double`

`TransmitterGain` — Transmitter antenna gain
`0` (default) | finite numeric scalar

Transmitter antenna gain, specified as a finite numeric scalar. Specify this value in dB.

Data Types: `double`

`ReceiverRange` — Packet reception range of node
`100` (default) | finite positive scalar

Packet reception range of node, specified as a finite positive scalar. Specify this value in meters. If a receiving node receives a signal from a sending node located beyond this value, the receiving node drops the received signal. To reduce the processing complexity of the simulation, set this property to a smaller value.

Data Types: `double`

`ReceiverGain` — Receiver antenna gain
`0` (default) | finite numeric scalar

Receiver antenna gain, specified as a finite numeric scalar. Specify this value in dB.

Data Types: `double`

`ReceiverSensitivity` — Receiver sensitivity
`-100` (default) | finite numeric scalar

Receiver sensitivity, specified as a finite numeric scalar. Specify this value in dBm. This property sets the minimum received power to detect the incoming signal. If the received power of an incoming signal is below this value, the node considers the signal invalid.

Data Types: `double`

`NoiseFigure` — Noise figure
`0` (default) | finite nonnegative numeric scalar

Noise figure, specified as a finite nonnegative numeric scalar. Specify this value in dB. Use this value to add thermal noise to the received signal.

Data Types: `double`

`InterferenceFidelity` — Fidelity to model interference
`0` (default) | `1`

Fidelity to model interference, specified as an integer in the range [0,1]. To consider packets overlapping in both time and frequency as interference, set this value to 0. To consider all the packets overlapping in time as interference, irrespective of frequency overlap, set this value to 1.

Data Types: `double`

`AdvertisingInterval` — Advertising interval
`0.02` (default) | scalar in the range [0.02, 10485.759375]

Advertising interval, specified as a scalar in the range [0.02, 10485.759375]. Specify this value in seconds. You must set this value as an integer multiple of 0.625 milliseconds. This value specifies the interval of an advertising event during which the transmission of advertising packets occurs.

Data Types: `double`

`RandomAdvertising` — Random advertising channel selection

`0` or `false` (default) | `1` or `true`

Random advertising channel selection, specified as `0` (`false`) or `1` (`true`). To model the random selection of advertising channels, set this value to `true`.

Data Types: `logical`

`ScanInterval` — Scan interval

`0.005` (default) | scalar in the range [0.005, 40.960]

Scan interval, specified as a scalar in the range [0.005, 40.960]. Specify this value in seconds. You must set this value as an integer multiple of 0.625 milliseconds. This value specifies the interval in which the node listens for the advertising packets.

Data Types: `double`

`MeshConfig` — Bluetooth mesh profile configuration parameters

`bluetoothMeshProfileConfig` object

Bluetooth mesh profile configuration parameters, specified as a `bluetoothMeshProfileConfig` object.

Dependencies

To enable this property, set the Role property to `"broadcaster-observer"`.

`ID` — Node identifier

scalar integer

This property is read-only.

Node identifier, returned as an integer. This value specifies a unique identifier for the node in the simulation.

Data Types: `double`

`Role` — Role of the Bluetooth LE Node

`"central"` | `"peripheral"` | `"isochronous-broadcaster"` | `"synchronized-receiver"` | `"broadcaster-observer"`

This property is read-only.

Role of the Bluetooth LE node, specified as one of these values.

Role Value	Description
`"central"`	Simulate Bluetooth LE node scenario
`"peripheral"`	Simulate Bluetooth LE node scenario
`"isochronous-broadcaster"`	Simulate Bluetooth LE broadcast audio network
`"synchronized-receiver"`	Simulate Bluetooth LE broadcast audio network
`"broadcaster-observer"`	Simulate Bluetooth LE mesh network

Data Types: `char` | `string`

`ConnectionConfig` — Bluetooth LE link layer (LL) connection configuration parameters
`bluetoothLEConnectionConfig` object

This property is read-only.

Bluetooth LE LL connection configuration parameters, returned as a `bluetoothLEConnectionConfig` object.

Dependencies

To enable this property, set the Role property to `"central"` or `"peripheral"`.

`PeripheralCount` — Number of Peripheral nodes associated with Central node
`0` (default) | integer

This property is read-only.

Number of Peripheral nodes associated with Central node, returned as an integer.

Dependencies

To enable this property, set the Role property to `"central"`.

Data Types: `double`

`BIGConfig` — Bluetooth LE broadcast isochronous group (BIG) configuration parameters
`bluetoothLEBIGConfig` object

This property is read-only.

Bluetooth LE BIG configuration parameters, returned as a `bluetoothLEBIGConfig` object.

Dependencies

To enable this property, set the Role property to `"isochronous-broadcaster"` or `"synchronized-receiver"`.

`FriendshipConfig` — Bluetooth mesh friendship configuration object for Friend and low power node (LPN)
`bluetoothMeshFriendshipConfig` object

This property is read-only.

Friendship configuration object for Friend and LPN, returned as a `bluetoothMeshFriendshipConfig` object.

Dependencies

To enable this property, set the Role property to `"broadcaster-observer"`.

Object Functions

Specific to This Object

addTrafficSource	Add data traffic source to Bluetooth LE node
channelInvokeDecision	(To be removed) Determine whether to apply channel to incoming signal
pushChannelData	(To be removed) Push data from channel to reception buffer

runNode	(To be removed) Run Bluetooth LE node
updateChannelList	Provide updated channel list to Bluetooth LE node
statistics	Get statistics of Bluetooth LE node

Examples

Create, Configure, and Simulate Bluetooth LE Network

This example shows you how to simulate a Bluetooth® low energy (LE) network by using Bluetooth® Toolbox.

Using this example, you can:

1. Create and configure a Bluetooth LE piconet with Central and Peripheral nodes.
2. Create and configure a link layer (LL) connection between Central and Peripheral nodes.
3. Add application traffic from the Central to Peripheral nodes.
4. Simulate Bluetooth LE network and retrieve the statistics of the Central and Peripheral nodes.

Create a wireless network simulator.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create a Bluetooth LE node, specifying the role as `"central"`. Specify the name and position of the node.

```
centralNode = bluetoothLENode("central");
centralNode.Name = "CentralNode";
centralNode.Position = [0 0 0];                 % In x-, y-, and z-coordinates in meters
```

Create a Bluetooth LE node, specifying the role as `"peripheral"`. Specify the name and position of the node.

```
peripheralNode = bluetoothLENode("peripheral");
peripheralNode.Name = "PeripheralNode";
peripheralNode.Position = [10 0 0]              % In x-, y-, and z-coordinates in meters
```

```
peripheralNode =
  bluetoothLENode with properties:

        TransmitterPower: 20
         TransmitterGain: 0
           ReceiverRange: 100
            ReceiverGain: 0
     ReceiverSensitivity: -100
             NoiseFigure: 0
    InterferenceFidelity: 0
                    Name: "PeripheralNode"
                Position: [10 0 0]

   Read-only properties:
                    Role: "peripheral"
        ConnectionConfig: [1x1 bluetoothLEConnectionConfig]
          TransmitBuffer: [1x1 struct]
                      ID: 2
```

Create a default Bluetooth LE configuration object to share the LL connection between the Central and Peripheral nodes.

```
cfgConnection = bluetoothLEConnectionConfig;
```

Specify the connection interval and connection offset. Throughout the simulation, the object establishes LL connection events for the duration of each connection interval. The connection offset is from the beginning of the connection interval.

```
cfgConnection.ConnectionInterval = 0.01; % In seconds
cfgConnection.ConnectionOffset = 0;      % In seconds
```

Specify the active communication period after the connection event is established between the Central and Peripheral nodes.

```
cfgConnection.ActivePeriod = 0.01 % In seconds
```

```
cfgConnection =
  bluetoothLEConnectionConfig with properties:

    ConnectionInterval: 0.0100
         AccessAddress: "5DA44270"
          UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
             Algorithm: 1
          HopIncrement: 5
     CRCInitialization: "012345"
    SupervisionTimeout: 1
               PHYMode: "LE1M"
         InstantOffset: 6
      ConnectionOffset: 0
          ActivePeriod: 0.0100
```

Configure the connection between Central and Peripheral nodes by using the `configureConnection` object function of the `bluetoothLEConnectionConfig` object.

```
configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Specify the data rate in kb/s and the packet size in bytes. Enable packet generation to generate an application packet with a payload.

```
traffic = networkTrafficOnOff(DataRate=100, ...
                              PacketSize=10, ...
                              GeneratePacket=true);
```

Add application traffic from the Central to the Peripheral node by using the `addTrafficSource` object function.

```
addTrafficSource(centralNode,traffic,"DestinationNode",peripheralNode.Name);
```

Create a Bluetooth LE network consisting of a Central and a Peripheral node.

```
nodes = {centralNode peripheralNode};
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,nodes)
```

Set the simulation time in seconds and run the simulation.

```
simulationTime = 0.5;
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve application, link layer (LL), and physical layer (PHY) statistics corresponding to the broadcaster and receiver nodes. For more information about the statistics, see “Bluetooth LE Node Statistics”.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
    Name: "CentralNode"
      ID: 1
     App: [1x1 struct]
      LL: [1x1 struct]
     PHY: [1x1 struct]
```

```
peripheralStats = statistics(peripheralNode)
```

```
peripheralStats = struct with fields:
    Name: "PeripheralNode"
      ID: 2
     App: [1x1 struct]
      LL: [1x1 struct]
     PHY: [1x1 struct]
```

More About

Events

Events are occurrences that the object triggers in response to an action such as a change in a property value or a user interaction with the application program. Listeners execute functions when notified that the event of interest occurs. Listeners respond by executing a callback function that has at least two input arguments defined, the event source object handle and the event data. For more information about events and listeners, see “Overview Events and Listeners” and “Event and Listener Concepts”.

The `bluetoothLENode` object defines these events.

`PacketTransmissionStarted`

When a node starts transmitting a packet, the object triggers the `PacketTransmissionStarted` event. This event passes an event notification and this structure as event data to the registered callback.

Field	Value	Description
`NodeName`	character vector	Node name, specifying the Name property of the Bluetooth LE node that starts the packet transmission

Field	Value	Description
`NodeID`	scalar positive integer	Unique node identifier, specifying the ID property of the Bluetooth LE node that starts the packet transmission
`CurrentTime`	scalar positive integer	Current time of the simulation in seconds
`PDU`	binary column vector	Protocol data unit (PDU) bits to be transmitted
`AccessAddress`	character vector representing a 4-octet hexadecimal number	Access address of the Bluetooth LE packet
`ChannelIndex`	integer in the range [0, 39]	Channel index for transmission
`PHYMode`	`"LE1M"`, `"LE2M"`, `"LE500K"`, or `"LE125K"`	Physical layer (PHY) transmission mode
`TransmittedPower`	scalar	Transmit power in dBm
`PacketDuration`	scalar positive integer	Packet duration in seconds.

`PacketReceptionEnded`

When the reception of a packet ends, the object triggers the `PacketReceptionEnded` event. This event passes an event notification and this structure as event data to the registered callback.

Field	Value	Description
`NodeName`	character vector	Node name, specifying the Name property of the Bluetooth LE node that receives the packet
`NodeID`	scalar positive integer	Unique node identifier, specifying the ID property of the Bluetooth LE node that receives the packet
`CurrentTime`	scalar positive integer	Current time of the simulation in seconds
`SourceNode`	character vector	Name of the source node, specifying the Name property of the Bluetooth LE node that starts the packet transmission
`SourceID`	scalar positive integer	Unique node identifier, specifying the ID property of the Bluetooth LE node that starts the packet transmission
`SuccessStatus`	logical scalar	Flag indicating the success status of the packet
`PDU`	binary column vector	PDU bits received
`AccessAddress`	character vector representing a 4-octet hexadecimal number	Access address of the packet

Field	Value	Description
`ChannelIndex`	integer in the range [0, 39]	Channel index for reception
`PHYMode`	`"LE1M"`, `"LE2M"`, `"LE500K"`, or `"LE125K"`	PHY reception mode
`ReceivedPower`	scalar	Received power in dBm
`SINR`	scalar	Signal-to-interference plus noise ratio in dB

`ChannelMapUpdated`

When the node starts using the updated channel map of the link, the object triggers the `ChannelMapUpdated` event. This event passes an event notification and this structure as event data to the registered callback.

Field	Value	Description
`NodeName`	character vector	Node name, specifying the Name property of the Bluetooth LE node at which the channel map is updated
`NodeID`	scalar positive integer	Unique node identifier, specifying the ID property of the Bluetooth LE node at which the channel map is updated
`CurrentTime`	scalar positive integer	Current time of the simulation in seconds
`PeerNode`	character vector	Name of the peer node, specifying the Name property of the Bluetooth LE at which the channel map is updated
`PeerID`	scalar positive integer	Unique node identifier of the peer node, specifying the ID property of the Bluetooth LE node at which the channel map is updated
`UpdatedChannelList`	vector of integers in the range [0, 36]	List of good channels

`AppDataReceived`

When there is data for the application from the node, the object triggers the `AppDataReceived` event. This event passes an event notification and this structure as event data to the registered callback.

Field	Value	Description
`NodeName`	character vector	Node name, specifying the Name property of the Bluetooth LE node at which the application data is destined

Field	Value	Description
`NodeID`	scalar positive integer	Unique node identifier, specifying the ID property of the Bluetooth LE node at which the application data is destined
`CurrentTime`	scalar positive integer	Current time of the simulation in seconds
`SourceNode`	character vector	Name of the source node, specifying the Name property of the Bluetooth LE node whose application data is received.
`ReceivedData`	vector of integers in the range [0, 255]	Received application data in decimal bytes

`MeshAppDataReceived`

When application data is received for a mesh node, the object triggers the `MeshAppDataReceived` event. This event passes an event notification and this structure as event data to the registered callback.

Field	Value	Description
`NodeName`	character vector	Node name, specifying the Name property of the Bluetooth LE node at which the application data is destined
`NodeID`	scalar positive integer	Unique node identifier, specifying the ID property of the Bluetooth LE node at which the application data is destined
`CurrentTime`	scalar positive integer	Current time of the simulation in seconds
`Message`	vector of integers in the range [0, 255]	Received access message
`SourceAddress`	character vector representing a 2-octet hexadecimal number	Source address of the message, specifying the Name property of the Bluetooth LE node that is the destination for the application data
`DestinationAddress`	character vector representing a 2-octet hexadecimal number	Destination address of the message, specifying the Name property of the Bluetooth LE node at which the application data is destined

`ConnectionEventEnded`

When the connection event ends, the object triggers the `ConnectionEventEnded` event. This event passes an event notification and this structure as event data to the registered callback.

Field	Value	Description
`NodeName`	character vector	Node name, specifying the Name property of the Bluetooth LE node at which the connection event ends
`NodeID`	scalar positive integer	Unique node identifier, specifying the ID property of the Bluetooth LE node at which the connection event ends
`CurrentTime`	scalar positive integer	Current time of the simulation in seconds
`Counter`	scalar positive integer in the range [0, 65535]	Current connection event counter
`TransmittedPackets`	scalar nonnegative integer	Number of transmitted packets in the connection event
`ReceivedPackets`	scalar nonnegative integer	Number of received packets in the connection event
`CRCFailedPackets`	scalar nonnegative integer	Number of received packets with cyclic redundancy check (CRC) failure

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Core Specifications Working Group. "Bluetooth Core Specification" v5.3. https://www.bluetooth.com/.

See Also

Objects

`wirelessNetworkSimulator` | `bluetoothNode` | `bluetoothLEConnectionConfig` | `bluetoothLEBIGConfig` | `bluetoothMeshProfileConfig` | `bluetoothMeshFriendshipConfig` | `wirelessNetworkSimulator`

Topics

“Create, Configure, and Simulate Bluetooth LE Network”
“Create, Configure, and Simulate Bluetooth LE Broadcast Audio Network”
“Create, Configure and Simulate Bluetooth Mesh Network”
“Estimate Packet Delivery Ratio of LE Broadcast Audio in Residential Scenario”
“Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network”

bluetoothMeshFriendshipConfig

Bluetooth mesh friendship configuration parameters

Description

Use the `bluetoothMeshFriendshipConfig` object to set the mesh friendship configuration between a Friend node and a low power node (LPN) based on Bluetooth Mesh Profile v1.0.1 [3].

Creation

Syntax

```
cfgMeshFriendship = bluetoothMeshFriendhshipConfig
cfgMeshFriendship = bluetoothMeshFriendshipConfig(Name=Value)
```

Description

`cfgMeshFriendship = bluetoothMeshFriendhshipConfig` creates a default Bluetooth mesh friendship configuration object.

`cfgMeshFriendship = bluetoothMeshFriendshipConfig(Name=Value)` sets properties on page 2-90 by using one or more name-value arguments. For example, `PollTimeout=2`sets the timeout for terminating the friendship between Friend node and LPN to 2 seconds.

Properties

`ReceiveDelay` — Receive delay requested by LPN

`0.01` (default) | scalar in the range [0.01, 0.255]

Receive delay requested by the LPN, specified as a scalar in the range [0.01, 0.255]. Specify this value in seconds. Set this value as an integer multiple of 1 milliseconds. This value specifies the time interval between when the LPN sends a request to the friend node and when the LPN listens for a response from the friend node. . For more information about receive delay, see Section 3.6.5.3 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `double`

`ReceiveWindow` — Receive window supported by Friend node

`0.001` (default) | scalar in the range [0.001, 0.255]

Receive window supported by the Friend node, specified as a scalar in the range [0.001, 0.255]. Specify this value in seconds. Set this value as an integer multiple of 1 milliseconds. This value specifies the amount of time that the LPN listens for a response (data or update message) from the Friend node. For more information about receive window, see Section 3.6.5.4 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `double`

`PollTimeout` — Timeout to terminate friendship between Friend node and LPN

1 (default) | scalar in the range [1, 345599.9]

Timeout to terminate friendship between the Friend node and LPN, specified as a scalar in the range [1, 345599.9]. Specify this value in seconds. Set this value as multiples of 100 milliseconds. This value specifies the maximum time allowed between two consecutive requests sent by the LPN to the Friend node to terminate the friendship. If the Friend node receives no request from the LPN within this value, this object terminates the friendship between the LPN and Friend node. For more information about poll timeout, see Section 3.6.5.3 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `double`

Object Functions

Specific to This Object

configureFriendship Configure friendship between Friend node and LPN

Examples

Create and Configure Bluetooth Mesh Node with Friendship Properties

Create a Bluetooth mesh profile configuration object, specifying the element address and enabling the Friend feature of the Friend node.

```
meshFriendshipCfg = bluetoothMeshProfileConfig(ElementAddress="000A",Friend=true)
```

```
meshFriendshipCfg =
  bluetoothMeshProfileConfig with properties:

             ElementAddress: "000A"
                      Relay: 0
                     Friend: 1
                   LowPower: 0
       NetworkTransmissions: 1
    NetworkTransmitInterval: 0.0100
                        TTL: 127
```

Create a Bluetooth mesh profile configuration object for an LPN, specifying the element address and enabling the low power feature of the LPN.

```
meshLPNCfg = bluetoothMeshProfileConfig(ElementAddress="000F",LowPower=true)
```

```
meshLPNCfg =
  bluetoothMeshProfileConfig with properties:

             ElementAddress: "000F"
                      Relay: 0
                     Friend: 0
                   LowPower: 1
       NetworkTransmissions: 1
    NetworkTransmitInterval: 0.0100
                        TTL: 127
```

Create a Bluetooth mesh node with the friend feature enabled. Specify the role as `"broadcaster-observer"` and assign the mesh profile configuration.

```
meshFriendNode = bluetoothLENode("broadcaster-observer",MeshConfig=meshFriendshipCfg);
```

Create a Bluetooth mesh node with the low power feature enabled. Specify the role as `"broadcaster-observer"` and assign the mesh profile configuration.

```
meshLPN = bluetoothLENode("broadcaster-observer",MeshConfig=meshLPNCfg);
```

Create a default Bluetooth mesh friendship configuration object. Specify the poll timeout, receive window supported by the Friend node, and receive delay requested by the LPN.

```
friendshipConfig = bluetoothMeshFriendshipConfig;
friendshipConfig.PollTimeout = 3;                       % In seconds
friendshipConfig.ReceiveWindow = 0.180;                 % In seconds
friendshipConfig.ReceiveDelay = 0.05                    % In seconds
```

```
friendshipConfig =
  bluetoothMeshFriendshipConfig with properties:

     ReceiveDelay: 0.0500
    ReceiveWindow: 0.1800
      PollTimeout: 3
```

Configure friendship between the Friend node and LPN.

```
configureFriendship(friendshipConfig,meshFriendNode,meshLPN);
```

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "Bluetooth Mesh Profile". v1.0.1. https://www.bluetooth.com/.

See Also

Objects

`bluetoothLENode` | `bluetoothLEBIGConfig` | `bluetoothLEConnectionConfig` | `bluetoothMeshProfileConfig`

Topics

“Bluetooth Mesh Networking”
“Create, Configure and Simulate Bluetooth Mesh Network”

“Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network”
“Create, Configure, and Visualize Bluetooth Mesh Network”
“Bluetooth Mesh Flooding in Wireless Sensor Networks”
“Energy Profiling of Bluetooth Mesh Nodes in Wireless Sensor Networks”

bluetoothMeshProfileConfig

Bluetooth mesh profile configuration parameters

Description

Use the `bluetoothMeshProfileConfig` object to set mesh profile configuration parameters at a Bluetooth low energy (LE) node based on Bluetooth Mesh Profile v1.0.1 [3].

Creation

Syntax

```
cfgMesh = bluetoothMeshProfileConfig
cfgMesh = bluetoothMeshProfileConfig(Name=Value)
```

Description

`cfgMesh = bluetoothMeshProfileConfig` creates a default Bluetooth mesh profile configuration object.

`cfgMesh = bluetoothMeshProfileConfig(Name=Value)` sets properties on page 2-94 by using one or more name-value arguments. For example, `Relay=true` enables relay support at the mesh node.

Properties

`ElementAddress` — Unicast address of element in mesh node

`"0001"` (default) | 4-element character vector | string scalar | character vector of size *N*-by-4 | string vector of size *N*

Unicast address of the mesh node element, specified as a 4-element character vector or a string scalar denoting a 2-octet hexadecimal unicast address. If the mesh profile contains multiple elements, specify this property as a character vector of size *N*-by-4 or a string vector of size *N*, where *N* is the total number of elements in the mesh profile. Set this property to a unique value for each element in the mesh network. For more information about node element addresses, see Section 2.3.4 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `char` | `string`

`Relay` — Flag to enable the relay feature

`false` (default) | `true`

Flag to enable the relay feature, specified as `true` or `false`. Enable the relay feature for a mesh node by setting this property to `true`. For more information about the relay feature, see Section 2.3.11 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `logical`

`Friend` — Flag to enable the friend feature
`false` (default) | `true`

Flag to enable the friend feature, specified as `true` or `false`. Enable the friend feature for a mesh node by setting this property to `true`. For more information about the friend feature, see Section 2.3.11 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `logical`

`LowPower` — Flag to enable the low power feature
`false` (default) | `true`

Flag to enable the low power feature, specified as `true` or `false`. Enable the low power feature for a mesh node by setting this property to `true`. For more information about the low power feature, see Section 2.3.11 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `logical`

`NetworkTransmissions` — Number of network transmissions
`1` (default) | integer in the range [1, 8]

Number of network transmissions, specified as an integer in the range [1, 8]. This value specifies the number of times a mesh node transmits a network message. For more information about network transmissions, see Section 4.2.19.1 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `double`

`NetworkTransmitInterval` — Time interval between consecutive network transmissions
`0.01` (default) | scalar in the range [0.01, 0.320]

Time interval between consecutive network transmissions, specified as a scalar in the range [0.01, 0.320]. Specify this value in seconds. This value specifies the time interval between consecutive network messages that the mesh node transmits. Set this value as an integer multiple of 10 milliseconds. For more information about the network transmission interval, see Section 4.2.19.2 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `double`

`TTL` — Time to live (TTL) for message
`127` (default) | integer in the range [0, 127], excluding 1

TTL for the message, specified as an integer in the range [0, 127], excluding 1. This value specifies the maximum number of hops that the transmitted message can traverse in the mesh network. When no value is set for the transmitted message, as in the higher layers of the mesh profile, the `bluetoothMeshProfileConfig` object uses this value in the generated network message. For more information about TTL, see Section 4.2.7 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `double`

`RelayRetransmissions` — Number of relay retransmissions
`1` (default) | integer in the range [1, 8]

Number of relay retransmissions, specified as an integer in the range [1, 8]. This value specifies the number of times a node retransmits a relayed network message. For more information about relay retransmissions, see Section 4.2.20.1 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `double`

`RelayRetransmitInterval` — Time interval between two consecutive relay retransmissions
`0.01` (default) | scalar in the range [0.01, 0.320]

Time interval between two consecutive relay retransmissions, specified as a scalar in the range [0.01, 0.320]. Specify this value in seconds. This value specifies the time interval between the consecutive relayed messages that the mesh node retransmits. Set this value as an integer multiple of 10 milliseconds. For more information about the relay retransmission interval, see Section 4.2.20.2 of Bluetooth Mesh Profile v1.0.1 [3].

Data Types: `double`

Examples

Create and Configure Bluetooth Mesh Node

Create a default Bluetooth mesh profile configuration object.

```
meshCfg = bluetoothMeshProfileConfig
```

```
meshCfg =
  bluetoothMeshProfileConfig with properties:

            ElementAddress: "0001"
                     Relay: 0
                   Friend: 0
                  LowPower: 0
       NetworkTransmissions: 1
    NetworkTransmitInterval: 0.0100
                       TTL: 127
```

Specify the element address of the mesh node.

```
meshCfg.ElementAddress = "0002";
```

Enable the relay feature of the mesh node. Set the number of network transmissions.

```
meshCfg.Relay = true;
meshCfg.NetworkTransmissions = 3;
```

Create a Bluetooth LE mesh node, specifying the role as `"Broadcaster-Observer"`. Assign the mesh profile configuration to the mesh node.

```
meshNode = bluetoothLENode("broadcaster-observer");
meshNode.MeshConfig = meshCfg;
```

Specify the advertising interval and scan interval.

```
meshNode.AdvertisingInterval = 0.03; % In seconds
meshNode.ScanInterval = 0.02;        % In seconds
```

Specify the name and position of the mesh node.

```
meshNode.Name = "SampleRelayNode";
meshNode.Position = [0 0 0]        % In meters
```

```
meshNode =
  bluetoothLENode with properties:
```

```
       TransmitterPower: 20
        TransmitterGain: 0
          ReceiverRange: 100
           ReceiverGain: 0
    ReceiverSensitivity: -100
            NoiseFigure: 0
   InterferenceFidelity: 0
    AdvertisingInterval: 0.0300
      RandomAdvertising: 0
           ScanInterval: 0.0200
             MeshConfig: [1x1 bluetoothMeshProfileConfig]
                   Name: "SampleRelayNode"
               Position: [0 0 0]

  Read-only properties:
                   Role: "broadcaster-observer"
        FriendshipConfig: [1x1 bluetoothMeshFriendshipConfig]
          TransmitBuffer: [1x1 struct]
                      ID: 17
```

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "Bluetooth Mesh Profile". v1.0.1. https://www.bluetooth.com/.

See Also

Objects

`bluetoothLENode` | `bluetoothLEBIGConfig` | `bluetoothLEConnectionConfig` | `bluetoothMeshFriendshipConfig`

Topics

“Bluetooth Mesh Networking”
“Create, Configure and Simulate Bluetooth Mesh Network”
“Establish Friendship Between Friend Node and LPN in Bluetooth Mesh Network”
“Bluetooth Mesh Flooding in Wireless Sensor Networks”
“Energy Profiling of Bluetooth Mesh Nodes in Wireless Sensor Networks”

bluetoothNode

Bluetooth BR node

Description

Use the `bluetoothNode` object to create and configure a Bluetooth basic rate (BR) node.

Creation

Syntax

```
BRNode = bluetoothNode(Role)
BRNode = bluetoothNode(Role,Name=Value)
```

Description

`BRNode = bluetoothNode(Role)` creates a default Bluetooth BR node object for the specified role `Role`. The `Role` argument sets the value of the Role property.

`BRNode = bluetoothNode(Role,Name=Value)` sets properties on page 2-98 of the Bluetooth BR node object by using one or more optional name-value arguments. For example, `"central",NoiseFigure=5` sets the noise figure of the specified node to 5 dB.

Properties

`Name` — Name of Bluetooth BR node
character vector | string scalar

Name of the Bluetooth BR node, specified as a character vector or a string scalar.

Data Types: `char` | `string`

`Position` — Node position in 3-D Cartesian coordinates
`[0 0 0]` (default) | three-element row vector

Node position in 3-D Cartesian coordinates, specified as a three-element numeric row vector. This value specifies the position of the node in Cartesian *x*, *y*, and *z* coordinates. Units are in meters.

Data Types: `double`

`TransmitterGain` — Transmitter antenna gain
`0` (default) | finite numeric scalar

Transmitter antenna gain, specified as a finite numeric scalar. Units are in dB.

Data Types: `double`

`ReceiverGain` — Receiver antenna gain
`0` (default) | finite numeric scalar

Receiver antenna gain, specified as a finite numeric scalar. Units are in dB.

Data Types: `double`

`ReceiverSensitivity` — Receiver sensitivity
`-100` (default) | finite numeric scalar

Receiver sensitivity, specified as a finite numeric scalar. This property sets the minimum power that the receiver requires to detect the incoming packet. If the received power of an incoming packet is below this value, the node drops the packet. Units are in dBm.

Data Types: `double`

`NoiseFigure` — Noise figure
`0` (default) | nonnegative finite scalar

Noise figure, specified as a nonnegative finite scalar. The object uses this value to apply thermal noise on the received packet. Units are in dB.

Data Types: `double`

`InterferenceFidelity` — Fidelity level to model interference
`0` (default) | `1`

Fidelity level to model interference, specified as `0` or `1`.

- If you set this value to `0`, the object considers packets overlapping in time and frequency as interference (co-channel interference).
- If you set this value to `1`, the object considers all the packets (in the 2.4GHz ISM band) overlapping in time as interference, irrespective of the frequency overlap.

Data Types: `double`

`ID` — Node identifier
integer scalar

This property is read-only.

Node identifier, stored as an integer scalar. This value specifies a unique identifier for the node in the simulation. The object assigns this value incrementally, starting from `1`.

Data Types: `double`

`Role` — Role of Bluetooth BR Node
`"central"` | `"peripheral"`

This property is read-only.

Role of the Bluetooth BR node, specified as `"central"` or `"peripheral"`.

Data Types: `char` | `string`

`NodeAddress` — Address of Bluetooth BR node
12-element character vector | string scalar denoting 6-octet hexadecimal value

Address of the Bluetooth BR node, stored as a 12-element character vector or a string scalar denoting a 6-octet hexadecimal value. This value is unique to each Bluetooth BR node and is derived from the ID property.

Data Types: `char` | `string`

`ConnectionConfig` — Bluetooth BR connection configuration parameters
`bluetoothConnectionConfig` object

This property is read-only.

Bluetooth BR connection configuration parameters, stored as a `bluetoothConnectionConfig` object.

`NumConnections` — Number of Peripheral nodes associated with Central node
positive integer scalar

This property is read-only.

Number of Peripheral nodes associated with the Central node, stored as a positive integer scalar.

Dependencies

To enable this property, set the Role property to `"central"`.

Data Types: `double`

Object Functions

Specific to This Object

addTrafficSource	Add data traffic source to Bluetooth BR node on ACL connection
statistics	Get statistics of Bluetooth BR node
updateChannelList	Update channel list of Bluetooth BR node

Examples

Create, Configure, and Simulate Bluetooth BR Nodes

Initialize the wireless network simulator.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and the other with the `"peripheral"` role. Specify the position of the Peripheral node, in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between the Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure the connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode)
```

```
connection =
  bluetoothConnectionConfig with properties:
```

```
    CentralToPeripheralACLPacketType: "DH1"
    PeripheralToCentralACLPacketType: "DH1"
                        SCOPacketType: "None"
                  HoppingSequenceType: "Connection adaptive"
                         UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 ... ]
                         PollInterval: 40
                       InstantOffset: 240
                    TransmitterPower: 20
                   SupervisionTimeout: 32000

   Read-only properties:
                      CentralAddress: "D091BBE70001"
                    PrimaryLTAddress: 1
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
    GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Specify the simulation time, in seconds.

```
simulationTime = 0.3;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve the application, baseband, and physical layer (PHY) statistics corresponding to the Central and Peripheral nodes.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
        Name: "Node1"
          ID: 1
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

```
peripheralStats = statistics(peripheralNode)
```

```
peripheralStats = struct with fields:
        Name: "Node2"
          ID: 2
         App: [1x1 struct]
```

```
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

Schedule Updated Channel Map for Bluetooth BR Node

Initialize the wireless network simulator.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and other with the `"peripheral"` role. Specify the position of the Peripheral node in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode)
```

```
connection =
  bluetoothConnectionConfig with properties:

    CentralToPeripheralACLPacketType: "DH1"
    PeripheralToCentralACLPacketType: "DH1"
                       SCOPacketType: "None"
                HoppingSequenceType: "Connection adaptive"
                       UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 ... ]
                       PollInterval: 40
                      InstantOffset: 240
                   TransmitterPower: 20
                 SupervisionTimeout: 32000

   Read-only properties:
                     CentralAddress: "D091BBE70001"
                  PrimaryLTAddress: 1
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
    GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Schedule channel list update at the Central node at 0.05 seconds to use channels 0 to 40.

```
scheduleAction(networkSimulator,@(varargin) updateChannelList(centralNode, ...
    0:40,DestinationNode=peripheralNode),[],0.05);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Specify the simulation time in seconds.

```
simulationTime = 0.3;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve application, baseband, and physical layer (PHY) statistics corresponding to the Central and Peripheral nodes.

```
centralStats = statistics(centralNode);
peripheralStats = statistics(peripheralNode);
```

Version History

Introduced in R2022b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed May 22, 2022. https://www.bluetooth.com/.

[2] Bluetooth Core Specifications Working Group. "Bluetooth Core Specification" v5.3. https://www.bluetooth.com/.

See Also

Objects

`bluetoothConnectionConfig` | `bluetoothLENode` | `wirelessNetworkSimulator`

Topics

“Create, Configure, and Simulate Bluetooth BR Piconet”
“Simulate Multiple Bluetooth BR Piconets with ACL Traffic”
“Bluetooth Node Statistics”

bluetoothPhyConfig

Bluetooth BR/EDR PHY configuration parameters

Description

The `bluetoothPhyConfig` object sets properties to configure the Bluetooth physical layer (PHY).

Creation

Syntax

```
cfgPHY = bluetoothPhyConfig
cfgPHY = bluetoothPhyConfig(Name,Value)
```

Description

`cfgPHY = bluetoothPhyConfig` creates a default Bluetooth PHY configuration object, `cfg`.

`cfgPHY = bluetoothPhyConfig(Name,Value)` sets properties on page 2-104 by using one or more name-value pairs. Enclose each property name in quotes. For example, `('Mode','EDR3M')` sets the PHY transmission mode to 3 Mbps.

Properties

Note For more information about Bluetooth BR/EDR waveform generator properties and their respective values, see Volume 2, Part B, Sections 6 and 7 of the Bluetooth Core Specification [2].

`Mode` — PHY transmission mode

`'BR'` (default) | `'EDR2M'` | `'EDR3M'`

PHY transmission mode, specified as `'BR'`, `'EDR2M'`, or `'EDR3M'`. This value indicates the type of Bluetooth BR/EDR waveform.

Data Types: `char` | `string`

`DeviceAddress` — Bluetooth BR/EDR device address

`'0123456789AB'` (default) | 12-element character vector | string scalar denoting 6-octet hexadecimal value

Bluetooth BR/EDR device address, specified as a 12-element character vector or a string scalar denoting a 6-octet hexadecimal value.

Data Types: `char` | `string`

`ModulationIndex` — Modulation Index

`0.32` (default) | scalar in the range [0.28, 0.35]

Modulation index, specified as a scalar in the range [0.28, 0.35]. This property is the modulation index that the object uses when performing Gaussian frequency shift keying (GFSK) modulation or demodulation.

Data Types: `double`

`SamplesPerSymbol` — Samples per symbol
8 (default) | positive integer

Samples per symbol, specified as a positive integer. The object uses this value for GFSK modulation and demodulation.

Data Types: `double`

`WhitenStatus` — Data whiten status
`'On'` (default) | `'Off'`

Data whiten status, specified as `'On'` or `'Off'`. To perform whitening on header and payload bits, set this value to `'On'`.

Data Types: `char` | `string`

`WhitenInitialization` — Whiten initialization
`[1; 1; 1; 1; 1; 1; 1]` (default) | 7-bit binary column vector

Whiten initialization, specified as a 7-bit binary column vector.

Dependencies

To enable this property, set the “WhitenStatus” on page 2-0 property to `'On'`.

Data Types: `double`

Note From R2022a, this object uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Examples

Create Bluetooth BR/EDR PHY Configuration Objects

Create two unique Bluetooth BR/EDR PHY configuration objects: one for synchronous connection oriented (SCO) logical transport and the other for connectionless peripheral broadcast (CPB) logical transport.

Create a default Bluetooth BR/EDR PHY configuration object for an SCO logical transport.

```
cfgPHY = bluetoothPhyConfig
```

```
cfgPHY =
  bluetoothPhyConfig with properties:

                    Mode: 'BR'
           DeviceAddress: '0123456789AB'
         ModulationIndex: 0.3200
        SamplesPerSymbol: 8
```

```
            WhitenStatus: 'On'
    WhitenInitialization: [7x1 double]
```

Create another Bluetooth BR/EDR PHY configuration object for a CSB logical transport by disabling the whiten status.

```
cfgPHY = bluetoothPhyConfig('WhitenStatus','Off')
```

```
cfgPHY =
  bluetoothPhyConfig with properties:

                Mode: 'BR'
       DeviceAddress: '0123456789AB'
     ModulationIndex: 0.3200
    SamplesPerSymbol: 8
        WhitenStatus: 'Off'
```

Version History

Introduced in R2020a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Properties must be specified as coder.Constant().

See Also

Functions

`bluetoothWaveformGenerator` | `bluetoothIdealReceiver` | `bleWaveformGenerator` | `bleIdealReceiver`

Objects

`bluetoothWaveformConfig`

bluetoothRangeConfig

Bluetooth BR/EDR or LE range estimation configuration parameters

Description

The `bluetoothRangeConfig` object parameterizes the `bluetoothRange` function for estimating the range between two Bluetooth basic rate/enhanced data rate (BR/EDR) or low energy (LE) devices.

Creation

Syntax

```
cfgRange = bluetoothRangeConfig
cfgRange = bluetoothRangeConfig(Name=Value)
```

Description

`cfgRange = bluetoothRangeConfig` creates a default Bluetooth BR/EDR or LE range estimation configuration object.

`cfgRange = bluetoothRangeConfig(Name=Value)` sets properties on page 2-107 by using one or more optional name-value arguments. For example, `bluetoothRangeConfig(Environment="Home")` sets the signal propagation environment to Home.

Properties

`Environment` — Signal propagation environment
`"Outdoor"` (default) | `"Industrial"` | `"Home"` | `"Office"`

Signal propagation environment, specified as `"Outdoor"`, `"Industrial"`, `"Home"`, or `"Office"`.

Data Types: `char` | `string`

`SignalPowerType` — Type of received signal power
`"ReceiverSensitivity"` (default) | `"ReceivedSignalPower"`

Type of received signal power, specified as `"ReceiverSensitivity"` or `"ReceivedSignalPower"`

Data Types: `char` | `string`

`ReceivedSignalPower` — Received signal power
`-79` (default) | negative scalar

Received signal power, specified as a negative scalar. Units are in dBm.

Dependencies

To enable this property, set the SignalPowerType property to `"ReceivedSignalPower"`.

Data Types: `double`

`Mode` — PHY transmission mode
`"LE1M"` (default) | `"LE2M"` | `"LE500K"` | `"LE125K"` | `"BR"` | `"EDR2M"` | `"EDR3M"`

Physical layer (PHY) transmission mode, specified as `"LE1M"`, `"LE2M"`, `"LE500K"`, `"LE125K"`, `"BR"`, `"EDR2M"`, or `"EDR3M"`.

Data Types: `char` | `string`

`ReceiverSensitivity` — Minimum signal strength that receiver can detect
`-94` (default) | negative scalar

Minimum signal strength that receiver can detect, specified as a negative scalar. Units are in dBm. This table shows the valid range of values of this property corresponding to the Mode property.

Mode Value	ReceiverSensitivity Range
• `"LE1M"` • `"LE2M"` • `"BR"` • `"EDR2M"` • `"EDR3M"`	[-110, -70]
`"LE125K"`	[-110, -82]
`"LE500K"`	[-110, -75]

Dependencies

To enable this property, set the SignalPowerType property to `"ReceiverSensitivity"`.

Data Types: `double`

`LinkMargin` — Link margin
`15` (default) | nonnegative scalar

Link margin, specified as nonnegative scalar. Units are in dB. This property specifies the difference between the minimum expected power received at the receiver's end, and receiver sensitivity.

Dependencies

To enable this property, set the SignalPowerType property to `"ReceiverSensitivity"`.

Data Types: `double`

`TransmitterPower` — Transmitter output power
`0` (default) | scalar in the range [-20, 20]

Transmitter output power, specified as a scalar in the range [-20, 20]. Units are in dBm.

Data Types: `double`

`TransmitterAntennaGain` — Transmitter antenna gain
`0` (default) | scalar in the range [-10, 10]

Transmitter antenna gain, specified as a scalar in the range [-10, 10]. Units are in dBi.

Data Types: `double`

`ReceiverAntennaGain` — Receiver antenna gain
`0` (default) | scalar in the range [-10, 10]

Receiver antenna gain, specified as a scalar in the range [-10, 10]. Units are in dBi.

Data Types: `double`

`TransmitterCableLoss` — Transmitter cable loss
`1.25` (default) | nonnegative scalar

Transmitter cable loss, specified as a nonnegative scalar. Units are in dB.

Data Types: `double`

`TransmitterAntennaHeight` — Transmitter antenna height
`1` (default) | positive scalar

Transmitter antenna height, specified as a positive scalar. Units are in meters.

Dependencies

To enable this property, set the Environment property to `"Outdoor"`.

Data Types: `double`

`ReceiverAntennaHeight` — Receiver antenna height
`1` (default) | positive scalar

Receiver antenna height, specified as a positive scalar. Units are in meters.

Dependencies

To enable this property, set the Environment property to `"Outdoor"`.

Data Types: `double`

`PathLossExponent` — Path loss exponent
`2.2` (default) | positive scalar

Path loss exponent, specified as a positive scalar. This property denotes the rate at which the received signal strength decreases as a function of the distance between the transmitter and receiver.

Dependencies

To enable this property, set the Environment property to `"Industrial"`.

Data Types: `double`

`StandardDeviation` — Standard deviation
`2.667` (default) | positive scalar

Standard deviation, specified as a positive scalar. Units are in dB.

Dependencies

To enable this property, set the Environment property to `"Industrial"`.

Data Types: `double`

`FSPLDistance` — Distance estimated by using free space path loss model
positive scalar

This property is read-only.

Distance estimated by using free space path loss model, returned as a positive scalar. Units are in meters.

Data Types: `double`

`PathLossModel` — Path loss model based on specified environment
`"TwoRayGroundReflection"` | `"LogNormalShadowing"` | `"NISTPAP02Task6"`

This property is read-only.

Path loss model based on the specified environment, returned as `"TwoRayGroundReflection"`, `"LogNormalShadowing"`, or `"NISTPAP02Task6"`. This table shows how this property depends on the Environment property.

`Environment` Value	`PathLossModel` Value
• `"Home"` • `"Office"`	`"NISTPAP02Task6"`
`"Outdoor"`	`"TwoRayGroundReflection"`
`"Industrial"`	`"LogNormalShadowing"`

Data Types: `char` | `string`

Object Functions

Specific to This Object

pathLoss Compute path loss and received signal power

Examples

Create Bluetooth Range Estimation Configuration Object

Create a default Bluetooth BR/EDR or LE range estimation configuration object.

```
cfgRange1 = bluetoothRangeConfig
```

```
cfgRange1 =
  bluetoothRangeConfig with properties:

                Environment: 'Outdoor'
           SignalPowerType: 'ReceiverSensitivity'
                      Mode: 'LE1M'
        ReceiverSensitivity: -94
                 LinkMargin: 15
           TransmitterPower: 0
     TransmitterAntennaGain: 0
        ReceiverAntennaGain: 0
```

```
        TransmitterCableLoss: 1.2500
           ReceiverCableLoss: 1.2500
    TransmitterAntennaHeight: 1
       ReceiverAntennaHeight: 1

   Read-only properties:
                FSPLDistance: 65.3645
               PathLossModel: 'TwoRayGroundReflection'
```

Set the PHY transmission mode to "EDR2M".

```
cfgRange1.Mode = "EDR2M";
```

Set an industrial environment for signal propagation and specify the path loss exponent.

```
cfgRange1.Environment = "Industrial";
cfgRange1.PathLossExponent = 3
```

```
cfgRange1 =
  bluetoothRangeConfig with properties:

                Environment: 'Industrial'
           SignalPowerType: 'ReceiverSensitivity'
                      Mode: 'EDR2M'
       ReceiverSensitivity: -94
                 LinkMargin: 15
           TransmitterPower: 0
    TransmitterAntennaGain: 0
       ReceiverAntennaGain: 0
      TransmitterCableLoss: 1.2500
         ReceiverCableLoss: 1.2500
          PathLossExponent: 3
         StandardDeviation: 2.6670

   Read-only properties:
              FSPLDistance: 65.3645
             PathLossModel: 'LogNormalShadowing'
```

Now, create another Bluetooth BR/EDR or LE range estimation configuration object, specifying the PHY transmission mode and receiver sensitivity as "LE500K" and -80, respectively. Specify the antenna height of transmitter and receiver.

```
cfgRange2 = bluetoothRangeConfig(Mode="LE500K",ReceiverSensitivity=-80,TransmitterAntennaHeight=1
```

```
cfgRange2 =
  bluetoothRangeConfig with properties:

                Environment: 'Outdoor'
           SignalPowerType: 'ReceiverSensitivity'
                      Mode: 'LE500K'
       ReceiverSensitivity: -80
                 LinkMargin: 15
           TransmitterPower: 0
    TransmitterAntennaGain: 0
       ReceiverAntennaGain: 0
      TransmitterCableLoss: 1.2500
         ReceiverCableLoss: 1.2500
```

```
    TransmitterAntennaHeight: 1.5000
       ReceiverAntennaHeight: 1.5000

   Read-only properties:
                FSPLDistance: 13.0419
               PathLossModel: 'TwoRayGroundReflection'
```

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bluetoothRange`

Topics

“Bluetooth LE Direction Finding for Tracking Node Position”

bluetoothRFPHYTestConfig

Bluetooth LE RF-PHY test configuration parameters

Description

Use the `bluetoothRFPHYTestConfig` object to set Bluetooth low energy (LE) radio frequency physical layer (RF-PHY) transmitter and receiver test configuration parameters compliant with RF-PHY.TS.p15 [2].

Creation

Syntax

```
cfgRFPHYTest = bluetoothRFPHYTestConfig
cfgRFPHYTest = bluetoothRFPHYTestConfig(Name=Value)
```

Description

`cfgRFPHYTest = bluetoothRFPHYTestConfig` creates a default Bluetooth LE RF-PHY test configuration object.

`cfgRFPHYTest = bluetoothRFPHYTestConfig(Name=Value)` sets properties on page 2-113 by using one or more optional name-value arguments. For example, `Test="Inband emissions"` sets the RF-PHY test operation to measure and verify inband emissions.

Properties

`Test` — RF-PHY test operation

`"Output Power"` (default) | `"Inband emissions"` | `"Modulation characteristics"` | `...`

RF-PHY test operation, specified as one of these values.

Test Value	Transmitter or Receiver Test	Description
`"Output Power"`	Transmitter	Measure and verify the maximum peak and average power emitted from the instrument under test (IUT).
`"Inband emissions"`	Transmitter	Measure and verify the inband spectral emissions at normal operating conditions.
`"Modulation characteristics"`	Transmitter	Measure and verify the modulation characteristics of the transmitted signal.

Test Value	Transmitter or Receiver Test	Description
`"Carrier frequency offset and drift"`	Transmitter	Measure and verify the carrier frequency offset and carrier drift of the transmitted signal.
`"Tx power stability"`	Transmitter	Measure and verify whether the angle of departure (AoD) transmit signal has settled at the beginning of the reference period and remains stable within the transmit slots.
`"C/I"`	Receiver	Measure and verify the receiver performance in the presence of co-channel and adjacent channel interference.
`"Blocking"`	Receiver	Measure and verify the receiver performance in the presence of interference sources operating outside the [2400, 2483.5] MHz band.
`"Intermodulation"`	Receiver	Measure and verify the intermodulation performance of the receiver.
`"Receiver sensitivity"`	Receiver	Measure and verify receiver sensitivity for receiving non-ideal signals at normal operating conditions.
`"Maximum input signal level"`	Receiver	Measure and verify the ability of a receiver to demodulate a wanted signal at high signal input levels.
`"PER report integrity"`	Receiver	Measure and verify the device under test (DUT) packet error rate (PER) report mechanism for the correct number of packets received at the tester with uncoded data.
`"IQ samples coherency"`	Receiver	Measure and verify the relative phase values derived from the in-phase (I) and quadrature (Q) values sampled on an IUT AoD or angle of arrival (AoA) receiver from a constant tone extension (CTE).

Test Value	Transmitter or Receiver Test	Description
`"IQ samples dynamic range"`	Receiver	Measure and verify I and Q values sampled on receipt of an AoD/AoA CTE from a peer device when the dynamic range of the CTE varies. This test marks any invalid samples as invalid.

Data Types: `char` | `string`

Mode — PHY transmission mode
`"LE1M"` (default) | `"LE2M"` | `"LE125K"` | `"LE500K"`

PHY transmission mode, specified as one of these values.

Mode Value	Characteristics
`"LE1M"`	• LE uncoded PHY • Modulation scheme is Gaussian frequency-shift keying (GFSK) • Data rate is 1 Mb/s
`"LE2M"`	• LE uncoded PHY • Modulation scheme is GFSK • Data rate is 2 Mb/s
`"LE125K"`	• LE Coded PHY • Modulation scheme is GFSK • Data rate is 125 kb/s
`"LE500K"`	• LE Coded PHY • Modulation scheme is GFSK • Data rate is 500 kb/s

This table shows the Mode property values supported by each Test property value.

Test Value	Mode value
`"Output Power"`	`"LE1M"` and `"LE2M"`
`"Inband emissions"`	`"LE1M"` and `"LE2M"`
`"Modulation characteristics"`	`"LE1M"`, `"LE2M"`, and `"LE125K"`
`"Carrier frequency offset and drift"`	`"LE1M"`, `"LE2M"`, and `"LE125K"`
`"Tx power stability"`	`"LE1M"` and `"LE2M"`
`"C/I"`	`"LE1M"`, `"LE2M"`, `"LE125K"`, and `"LE500K"`
`"Blocking"`	`"LE1M"` and `"LE2M"`
`"Intermodulation"`	`"LE1M"` and `"LE2M"`
`"Receiver sensitivity"`	`"LE1M"`, `"LE2M"`, `"LE125K"`, and `"LE500K"`
`"Maximum input signal level"`	`"LE1M"` and `"LE2M"`

Test Value	Mode value
"PER report integrity"	"LE1M", "LE2M", "LE125K", and "LE500K"
"IQ samples coherency"	"LE1M" and "LE2M"
"IQ samples dynamic range"	"LE1M" and "LE2M"

Data Types: char | string

PayloadLength — Payload length of Bluetooth LE test packet
37 (default) | integer in the range [31, 255]

Payload length of the Bluetooth LE test packet, in bytes, specified as an integer in the range [31, 255]. This value specifies the number of bytes that the object processes in a packet.

Dependencies

To enable this property, set the Test property to "Output Power", "Inband emissions", "Modulation characteristics", "Carrier frequency offset and drift", "C/I", "Blocking", "Intermodulation", "Receiver sensitivity", "Maximum input signal level", or "PER report integrity".

Data Types: double

PacketType — Type of Bluetooth LE test packet
"Disabled" (default) | "ConnectionCTE"

Type of Bluetooth LE test packet, specified as "ConnectionCTE" or "Disabled". This table shows the Mode values that each value of this property supports.

PacketType Value	Mode Value
"ConnectionCTE"	"LE1M" or "LE2M"
"Disabled"	"LE1M", "LE2M", "LE125K", or "LE500K"

Because the CTE field position is same for an LE test packet and the data packet, this object specifies the "ConnectionCTE" packet type for the CTE-based tests.

Dependencies

To enable this property, set the Test property to "Output power", "Carrier frequency offset and drift".

Data Types: char | string

CTELength — Length of CTE
2 (default) | integer in the range [2, 20]

Length of CTE, specified as an integer in the range [2, 20]. This value specifies the length of the CTE in 8 microsecond duration.

Dependencies

To enable this property, apply these configurations.

- Set the Test input to "Tx power stability", "IQ samples coherency", or "IQ samples dynamic range".

- Set the PacketType input to `"ConnectionCTE"`.

Data Types: `double`

`CTEType` — Type of CTE

`[0;0]` (default) | `[0;1]` | `[1;0]`

Type of CTE, specified as `[0;0]`, `[0;1]`, or `[1; 0]`. This table shows the valid Test property values for each value of this property.

CTEType Value	Characteristics	Valid Test Values
`[0;0]`	AoA	• `"Carrier frequency offset and drift"` • `"Output power"` • `"IQ samples coherency"` • `"IQ samples dynamic range"`
`[0;1]`	AoD with 2 µs slot duration	• `"Tx power stability"` • `"IQ samples coherency"` • `"IQ samples dynamic range"`
`[1;0]`	AoD with 1 µs slot duration	• `"Tx power stability"` • `"IQ samples coherency"` • `"IQ samples dynamic range"`

Dependencies

To enable this property, apply these configurations.

- Set the Test property to `"Tx power stability"`, `"Carrier frequency offset and drift"`, `"IQ samples coherency"`, or `"IQ samples dynamic range"`.
- Set the PacketType property to `"ConnectionCTE"`.

Data Types: `double`

`SamplesPerSymbol` — Samples per symbol

8 (default) | positive integer

Samples per symbol, specified as a positive integer.

Data Types: `double`

`ArraySize` — Size of antenna array

4 (default) | positive integer greater than or equal to 2 | two-element row vector of positive integers greater than or equal to 2

Size of antenna array, specified as a positive integer or a two-element row vector of positive integers. Setting this property to a positive integer specifies a uniform linear array (ULA) antenna array design with array elements on the *y*-axis. Setting this property to a vector specifies a uniform rectangular array (URA) antenna array design. The vector must be of the form [*r c*], where *r* and *c* denote the

number of row elements and column elements in the antenna array, respectively. In this case, the row elements and column elements are present along the *y*-axis and *z*-axis, respectively. You must specify a value greater than 2 for both ULA and URA antenna design.

Dependencies

To enable this property, set the Test property to `"Tx power stability"`, `"IQ samples coherency"`, or `"IQ samples dynamic range"`.

Data Types: `double`

`ElementSpacing` — Normalized element spacing with respect to signal wavelength

`0.5` (default) | positive number less than or equal to 0.5 | two element row vector of positive numbers less than or equal to 0.5

Normalized element spacing with respect to signal wavelength, specified as a positive number less than or equal to 0.5 or a 2-element row vector of positive numbers less than or equal to 0.5. Setting this property to a positive number specifies a ULA antenna array design with array elements on the *y*-axis. Setting this property to a vector specifies a URA antenna array design. The vector must be of the form [*sr sc*], where *sr* and *sc* denote the spacing between row and column elements of the antenna array, respectively. In this case, the rows and columns are present along the *y*-axis and *z*-axis, respectively.

Dependencies

To enable this property, set the Test property to `"Tx power stability"`, `"IQ samples coherency"`, or `"IQ samples dynamic range"`.

Data Types: `double`

`CenterFrequency` — Frequency of operation

`"Low"` (default) | `"Mid"` | `"High"`

Frequency of operation, specified as `"Low"`, `"Mid"`, or `"High"`. For more information about this property, see RF-PHY.TS.p15 Section 6 Table 6.1 [2].

Dependencies

To enable this property, set the Test property to `"Output Power"`, `"Inband emissions"`, `"Modulation characteristics"`, `"Carrier frequency offset and drift"`, `"Tx power stability"`, `"C/I"`, `"Intermodulation"`, `"Receiver sensitivity"`, `"Maximum input signal level"`, `"IQ samples coherency"`, or `"IQ samples dynamic range"`.

Data Types: `double`

`NumTestFrequencies` — Number of test frequencies

`10` (default) | positive integer

Number of test frequencies, specified as a positive integer. This table shows the Mode values supported by this property.

`NumTestFrequencies` Values	Mode Value
integer in the range [1, 78]	`"LE1M"`
integer in the range [1, 74]	`"LE2M"`

This property specifies the number of adjacent channels of the operating center frequency at which the object measures power at the transmitter and receiver end.

Dependencies

To enable this property, set the Test property to `"Inband emissions"`.

Data Types: `double`

`ResolutionBW` — Resolution bandwidth

`3×10^6` (default) | positive scalar

Resolution bandwidth, specified as a positive scalar in Hz.

Dependencies

To enable this property, set the Test property to `"Output power"`, `"Inband emissions"`, or `"Tx power stability"`.

Data Types: `double`

`OutputPower` — Transmitter output power

`0` (default) | scalar in the range [-20, 20]

Transmitter output power, specified as a scalar in the range [-20, 20]. Units are in dBm.

Dependencies

To enable this property, set the Test property to `"Output power"`, `"Inband emissions"`, or `"Tx power stability"`.

Data Types: `double`

`InitialFrequencyOffset` — Initial carrier frequency offset

`0` (default) | scalar in the range $[-100 \times 10^3, 100 \times 10^3]$

Initial carrier frequency offset, specified as a scalar in the range $[-100 \times 10^3, 100 \times 10^3]$. Units are in Hz.

Dependencies

To enable this property, set the Test property to `"Modulation characteristics"` or `"Carrier frequency offset and drift"`.

Data Types: `double`

`CarrierDrift` — Carrier frequency drift

`0` (default) | scalar in the range $[-50 \times 10^3, 50 \times 10^3]$

Carrier frequency drift, specified as a scalar in the range $[-50 \times 10^3, 50 \times 10^3]$. Units are in Hz.

Dependencies

To enable this property, set the Test property to `"Modulation characteristics"` or `"Carrier frequency offset and drift"`.

Data Types: `double`

`WantedSignalLevel` — Wanted signal input level

`-67` (default) | scalar | row vector of length 4

Wanted signal input level in dBm, specified as a scalar or a row vector of length 4. This property depends on the Test property values as follows.

- If you set the Test property to `"C/I"`, `"Blocking"`, `"Intermodulation"`, `"IQ samples coherency"`, `"Receiver sensitivity"`, `"Maximum input signal level"`, or `"PER report integrity"`, specify this property as a scalar.
- If you set the Test property to `"IQ samples dynamic range"`, specify this property as a row vector of size 4.

Dependencies

To enable this property, set the Test property to `"C/I"`, `"Blocking"`, `"Intermodulation"`, `"IQ samples coherency"`, `"IQ samples dynamic range"`, `"Receiver sensitivity"`, `"Maximum input signal level"`, or `"PER report integrity"`.

Data Types: `double`

`Interferer1SignalLevel` — Interference signal 1 input level

`[-88 82 -50 -40]` (default) | scalar | row vector of length 4

Interference signal 1 input level in dBm, specified as a scalar or a row vector of length 4. This property depends on the Test property values as follows.

- If you set the Test property to `"Intermodulation"`, specify this property as a scalar.
- If you set the Test property to `"C/I"`, specify this property as a row vector of size 4. The vector is based on the frequency offset from the operating CenterFrequency value *fRx*. Each element in the vector specifies the interference signal level at [*fRx* *fRx* + 1 *fRx* + 2 *fRx* + >= 3], where the frequency offset to *fRx* is in MHz. The interference signal is Bluetooth modulated.

Dependencies

To enable this property, set the Test property to `"Intermodulation"` or `"C/I"`.

Data Types: `double`

`Interferer2SignalLevel` — Interference signal 2 input level

`-50` (default) | scalar

Interference signal 2 input level in dBm, specified as a scalar. The interference signal is a sinusoidal unmodulated carrier.

Dependencies

To enable this property, set the Test property to `"Intermodulation"`.

Data Types: `double`

`BlockingSignalLevel` — Blocking signal input level

`[-30 -35 -35 -30]` (default) | vector of size [1, 4]

Blocking signal input level in dBm, specified as a vector of size [1, 4]. This table shows the frequency band corresponding to each index of this vector.

`BlockingSignalLevel` property vector index	Frequency band in MHz
1	[30, 2000]
2	[2003, 2399]
3	[2482, 2997]

BlockingSignalLevel property vector index	Frequency band in MHz
4	[3000, 12750]

Dependencies

To enable this property, set the Test property to `"Blocking"`.

Data Types: `double`

`BlockingSignalFreqResolution` — Frequency resolution of blocking signal

`[10 × 10^6 3 × 10^6 3 × 10^6 25 × 10^6]` (default) | positive vector of size [1, 4]

Frequency resolution of the blocking signal in Hz, specified as a positive vector of size [1, 4]. This table shows the frequency band corresponding to each index of this vector.

BlockingSignalFreqResolution property vector index	Frequency band in MHz
1	[30, 2000]
2	[2003, 2399]
3	[2482, 2997]
4	[3000, 12750]

Dependencies

To enable this property, set the Test property to `"Blocking"`.

Data Types: `double`

`NumPackets` — Number of Bluetooth LE test packets

`1` (default) | positive integer

Number of Bluetooth LE test packets, specified as a positive integer. This value specifies the number of test packets that the object runs.

Data Types: `double`

`PERUpperLimit` — Upper limit of PER

`30.8` (default) | positive scalar

Upper limit of packet error rate (PER), specified as a positive scalar that is a percentage. For more information about this property, see RF-PHY.TS.p15 Section 6 Table 6.4 [2].

To add

Dependencies

To enable this property, set the Test property to `"C/I"`, `"Blocking"`, `"Intermodulation"`, `"Receiver sensitivity"`, `"Maximum input signal level"`, or `"PER report integrity"`.

Data Types: `double`

`TestID` — RF-PHY test ID

character vector

This property is read-only.

RF-PHY test ID, returned as a character vector. This table shows the Test ID values that result from the Test, Mode, and CTEType values that you set.

TestID Value	Test Value	Mode Value	CTEType Value
'RF-PHY/TRM/BV-15-C'	"Output power"	"LE1M" and "LE2M"	"ConnectionCTE"
'RF-PHY/TRM/BV-01-C'			"Disabled"
'RF-PHY/TRM/BV-08-C'	"Inband emissions"	"LE2M"	"Disabled"
'RF-PHY/TRM/BV-03-C'		"LE1M"	
'RF-PHY/TRM/BV-09-C'	"Modulation characteristics"	"LE1M"	
'RF-PHY/TRM/BV-11-C'		"LE2M"	
'RF-PHY/TRM/BV-13-C'		"LE125K"	
'RF-PHY/TRM/BV-16-C'	"Carrier frequency offset and drift"	"LE1M"	"ConnectionCTE"
'RF-PHY/TRM/BV-06-C'			"Disabled"
'RF-PHY/TRM/BV-17-C'		"LE2M"	"ConnectionCTE"
'RF-PHY/TRM/BV-12-C'			"Disabled"
'RF-PHY/TRM/BV-14-C'		"LE125K"	"Disabled"
'RF-PHY/TRM/PS/BV-02-C'	"Tx power stability"	"LE1M" with CTEType value [1;0]	"Disabled"
'RF-PHY/TRM/PS/BV-01-C'		"LE1M" with CTEType value [0;1]	
'RF-PHY/TRM/PS/BV-04-C'		"LE2M" with CTEType value [1;0]	
'RF-PHY/TRM/PS/BV-03-C'		"LE2M" with CTEType value [0;1]	
'RF-PHY/RCV/BV-15-C'	"C/I"	"LE1M"	
'RF-PHY/RCV/BV-21-C'		"LE2M"	
'RF-PHY/RCV/BV-34-C'		"LE500K"	
'RF-PHY/RCV/BV-35-C'		"LE125K"	

TestID Value	Test Value	Mode Value	CTEType Value
'RF-PHY/RCV/BV-16-C'	"Blocking"	"LE1M"	"Disabled"
'RF-PHY/RCV/BV-22-C'		"LE2M"	
'RF-PHY/RCV/BV-17-C'	"Intermodulation"	"LE1M"	
'RF-PHY/RCV/BV-23-C'		"LE2M"	
'RF-PHY/RCV/BV-14-C'	"Receiver sensitivity"	"LE1M"	
'RF-PHY/RCV/BV-20-C'		"LE2M"	
'RF-PHY/RCV/BV-32-C'		"LE500K"	
'RF-PHY/RCV/BV-33-C'		"LE125K"	
'RF-PHY/RCV/BV-18-C'	"Maximum input signal level"	"LE1M"	"Disabled"
'RF-PHY/RCV/BV-24-C'		"LE2M"	
'RF-PHY/RCV/BV-19-C'	"PER report integrity"	"LE1M"	
'RF-PHY/RCV/BV-25-C'		"LE2M"	
'RF-PHY/RCV/BV-36-C'		"LE500K"	
'RF-PHY/RCV/BV-37-C'		"LE125K"	
'RF-PHY/RCV/IQC/BV-05-C'	"IQ samples coherency"	"LE1M" with CTEType value [0;0]	"ConnectionCTE"
'RF-PHY/RCV/IQC/BV-02-C'		"LE1M" with CTEType value [1;0]	
'RF-PHY/RCV/IQC/BV-01-C'		"LE1M" with CTEType value [0;1]	
'RF-PHY/RCV/IQDR/BV-06-C'		"LE2M" with CTEType value [0;0]	
'RF-PHY/RCV/IQC/BV-04-C'		"LE2M" with CTEType value [1;0]	
'RF-PHY/RCV/IQC/BV-03-C'		"LE2M" with CTEType value [0;1]	
'RF-PHY/RCV/IQDR/BV-11-C'	"IQ samples dynamic range"	"LE1M" with CTEType value [0;0]	"ConnectionCTE"

TestID Value	Test Value	Mode Value	CTEType Value
'RF-PHY/RCV/IQC/BV-08-C'		"LE1M" with CTEType value [1;0]	
'RF-PHY/RCV/IQC/BV-07-C'		"LE1M" with CTEType value [0;1]	
'RF-PHY/RCV/IQDR/BV-12-C'		"LE2M" with CTEType value [0;0]	
'RF-PHY/RCV/IQC/BV-10-C'		"LE2M" with CTEType value [1;0]	
'RF-PHY/RCV/IQC/BV-09-C'		"LE2M" with CTEType value [0;1]	

Data Types: char

FrequencySpan — Frequency span for test measurement

scalar

This property is read-only.

Frequency span for test measurement, returned as a scalar. Units are in Hz. If you set a positive value for this property, the ratio of this value to ResolutionBW property must be at least 2.

Dependencies

To enable this property, set the Test property to "Output power", "Inband emissions", or "Tx power stability". This table shows the frequency span values that result from the Test value that you set.

Test Value	FrequencySpan Value
"Output Power" or "Tx power stability"	0
"Inband emissions"	1×10^6

Data Types: double

ChannelFilterFrequencies — Channel filter frequencies

scalar

This property is read-only.

Channel filter frequencies, returned as a scalar. This property specifies the channel filter frequency response that the object uses for test measurement. Units are in Hz.

Dependencies

To enable this property, set the Test property to "Modulation characteristics" or "Carrier frequency offset and drift". This table shows the channel filter frequency values that result from the Mode value that you set.

Mode Value	ChannelFilterFrequencies Value
"LE2M"	[1100×10³ 1300×10³ 2×10⁶ 4×10⁶]
"LE1M" and "LE125K"	[550×10³ 650×10³ 1×10⁶ 2×10⁶]

Data Types: `double`

`ChannelFilterAmplitudes` — Channel filter amplitudes

`[0.25 -3 -14 -44]`

This property is read-only.

Channel filter amplitudes, returned as a real scalar. This property specifies the channel filter amplitude that the object uses for test measurement. Units are in dB.

Dependencies

To enable this property, set the Test property to `"Modulation characteristics"` or `"Carrier frequency offset and drift"`.

Data Types: `double`

`PayloadSequence` — Payload sequence of Bluetooth LE test packet

character array | cell array of two character arrays

This property is read-only.

Payload sequence of the Bluetooth LE test packet, returned as a character array or a cell array of two character arrays. This table shows the payload sequence values that result from the Test, Mode, and PacketType values that you set.

PayloadSequence Value	Test Value	Mode Value	PacketType Value
`'11111111'`	`"Modulation characteristics"`	`"LE125K"`	`"Disabled"`
`{'11110000','10101010'}`		`"LE1M"` and `"LE2M"`	`"ConnectionCTE"`, `"ConnectionlessCTE"`, and `"Disabled"`
`'11111111'`	`"Carrier frequency offset and drift"`	`"LE125K"`	`"Disabled"`
`'11110000'`		`"LE1M"` and `"LE2M"`	`"ConnectionCTE"`
`'10101010'`			`"Disabled"`
`'PRBS9'`	Others	`"LE1M"`, `"LE2M"`, `"LE125K"`, and `"LE500K"`	`"ConnectionCTE"` and `"Disabled"`

Dependencies

To enable this property, set the Test property to `"Output Power"`, `"Inband emissions"`, `"Modulation characteristics"`, `"Carrier frequency offset and drift"`, `"C/I"`, `"Blocking"`, `"Intermodulation"`, `"Receiver sensitivity"`, `"Maximum input signal level"`, or `"PER report integrity"`.

Data Types: `char`

`Interferer1PayloadSequence` — Payload sequence of interference signal 1

`'PRBS15'` (default)

This property is read-only.

Payload sequence of interference signal 1, returned as `"PRBS15"`. This property specifies the payload sequence that the object uses for the Bluetooth modulated interference signal.

Dependencies

To enable this property, set the Test property to `"Intermodulation"` or `"C/I"`.

Data Types: `char`

Examples

Create Bluetooth LE RF-PHY Test Configuration Object

Create a default Bluetooth LE RF-PHY test configuration object.

```
cfgRFPHYTest = bluetoothRFPHYTestConfig
```

```
cfgRFPHYTest =
  bluetoothRFPHYTestConfig with properties:

               Test: 'Output power'
               Mode: 'LE1M'
      PayloadLength: 37
         PacketType: 'Disabled'
    SamplesPerSymbol: 8

   Measurements:
     CenterFrequency: 'Low'
        ResolutionBW: 3000000
         OutputPower: 0
          NumPackets: 1

   Read-only:
            TestID: 'RF-PHY/TRM/BV-01-C'
     FrequencySpan: 0
     PayloadSequence: 'PRBS9'
```

Create a Bluetooth LE RF-PHY test configuration object, specifying the PHY transmission mode as LE125K. Set the RF-PHY test operation to measure receiver performance in co- and adjacent channel interference conditions. Set the wanted signal level to -75 dBm.

```
cfgRFPHYTest = bluetoothRFPHYTestConfig(Test='C/I', ...
    Mode='LE125K', ...
    WantedSignalLevel=-75)                                    % In dBm
```

```
cfgRFPHYTest =
  bluetoothRFPHYTestConfig with properties:

                         Test: 'C/I'
                         Mode: 'LE125K'
                PayloadLength: 37
              SamplesPerSymbol: 8

   Measurements:
              CenterFrequency: 'Low'
            WantedSignalLevel: -75
```

```
      Interferer1SignalLevel: [-88 -82 -50 -40]
                  NumPackets: 1
              PERUpperLimit: 30.8000

  Read-only:
                      TestID: 'RF-PHY/RCV/BV-35-C'
             PayloadSequence: 'PRBS9'
   Interferer1PayloadSequence: 'PRBS15'
```

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Radio Frequency Physical Layer (RF PHY)" RF-PHY.TS.p15. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bluetoothWaveformGenerator` | `bleWaveformGenerator` | `bluetoothTestWaveform` | `bluetoothIdealReceiver` | `bleIdealReceiver`

Objects

`bluetoothTestWaveformConfig` | `bluetoothWaveformConfig` | `bluetoothPhyConfig`

bluetoothTestWaveformConfig

Bluetooth BR/EDR or LE test waveform configuration parameters

Description

The `bluetoothTestWaveformConfig` object parameterizes the `bluetoothTestWaveform` function for generating the Bluetooth basic rate/enhanced data rate (BR/EDR) or low energy (LE) test waveform.

Creation

Syntax

```
cfgTestWaveform = bluetoothTestWaveformConfig
cfgTestWaveform = bluetoothTestWaveformConfig(Name=Value)
```

Description

`cfgTestWaveform = bluetoothTestWaveformConfig` creates a default Bluetooth BR/EDR or LE test waveform generation configuration object.

`cfgTestWaveform = bluetoothTestWaveformConfig(Name=Value)` sets properties on page 2-128 by using one or more optional name-value arguments. For example, `Mode="BR"` sets the physical layer (PHY) transmission mode to basic rate.

Properties

Mode — PHY transmission mode

`"LE1M"` (default) | `"LE2M"` | `"LE125K"` | `"LE500K"` | `"BR"` | `"EDR2M"` | `"EDR3M"`

PHY transmission mode, specified as one of these values.

Mode Value	Characteristics
`"BR"`	• Modulation scheme is Gaussian frequency-shift keying (GFSK) • Data rate is 1 Mb/s
`"EDR2M"`	• Modulation scheme is π/4 differential quadrature phase-shift keying (DQPSK) • Data rate is 2 Mb/s
`"EDR3M"`	• Modulation scheme is 8 differential phase-shift keying (DPSK) • Data rate is 3 Mb/s

Mode Value	Characteristics
"LE1M"	• LE uncoded PHY • GFSK • Data rate is 1 Mb/s
"LE2M"	• LE uncoded PHY • Modulation scheme is GFSK • Data rate is 2 Mb/s
"LE125K"	• LE Coded PHY • Modulation scheme is GFSK • Data rate is 125 kb/s
"LE500K"	• LE Coded PHY • Modulation scheme is GFSK • Data rate is 500 kb/s

Data Types: char | string

PayloadType — Type of payload for which to generate test packet

0 (default) | integer in the range [0, 7]

Type of payload for which to generate test packet, specified as an integer in the range [0, 7].

Data Types: double

PayloadLength — Payload length of Bluetooth LE test packet

255 (default) | integer in the range [0, 255]

Payload length of Bluetooth LE test packet, specified as an integer in the range [0, 255]. This value specifies the number of bytes that the object processes in a packet.

Dependencies

To enable this property, set the Mode property to "LE1M", "LE2M", "LE125K", or "LE500K".

Data Types: double

PacketType — Type of test packet

"Disabled" (default) | "ConnectionCTE" | "DH1" | ...

Type of test packet, specified as one of these values corresponding to the Mode property.

Mode Value	PacketType Value
"BR"	• "DH1" • "DH3" • "DH5" • "DM1" • "DM3" • "DM5"

Mode Value	PacketType Value
"EDR2M"	• "2-DH1" • "2-DH3" • "2-DH5" • "2-EV3" • "2-EV5"
"EDR3M"	• "3-DH1" • "3-DH3" • "3-DH5" • "3-EV3" • "3-EV5"
"LE1M" or "LE2M"	• "ConnectionCTE" • "Disabled"
"LE125K" or "LE500K"	• "Disabled"

Because the constant tone extension (CTE) field position is the same for an LE test packet and the data packet, this object specifies the "ConnectionCTE" packet type for the CTE-based tests.

Data Types: char | string

FixedPayloadLength — Fixed payload length of Bluetooth BR/EDR test packet
positive integer

This property is read-only.

Fixed payload length of Bluetooth BR/EDR test packet, returned as a positive integer. Units are in bytes.

Dependencies

To enable this property, set the Mode property to "BR", "EDR2M", or "EDR3M". This table shows the valid values of this property for each PacketType of the "BR", "EDR2M", and "EDR3M" mode.

Value of Mode	Value of PacketType	Value of FixedPayloadLength
"BR"	"DH1"	27
	"DH3"	183
	"DH5"	339
	"DM1"	17
	"DM3"	121
	"DM5"	224
"EDR2M"	"2-DH1"	54
	"2-DH3"	367
	"2-DH5"	679
	"2-EV3"	60

Value of Mode	Value of PacketType	Value of FixedPayloadLength
	"2-EV5"	360
"EDR3M"	"3-DH1"	83
	"3-DH3"	552
	"3-DH5"	1021
	"3-EV3"	90
	"3-EV5"	540

Data Types: double

SamplesPerSymbol — Samples per symbol

8 (default) | positive integer

Samples per symbol, specified as a positive integer.

Data Types: double

WhitenStatus — Data whiten status

"On" (default) | "Off"

Data whiten status, specified as "On" or "Off". If you set this value to "On", this object performs whitening on the header and payload.

Data Types: char | string

WhitenInitialization — Whiten initialization

[1; 1; 1; 1; 1; 1; 1] (default) | 7-bit binary-valued column vector

Whiten initialization, specified as a 7-bit binary-valued column vector. This value specifies the shift register initialization for data whitening or dewhitening.

Dependencies

To enable this property, set the WhitenStatus property to "On".

Data Types: double

ModulationIndex — Modulation index

0.32 (default) | scalar in the range [0.28, 0.35]

Modulation index, specified as a scalar in the range [0.28, 0.35]. The object uses this value to perform GFSK modulation or demodulation.

Dependencies

To enable this property, set the Mode property to "BR", "EDR2M", or "EDR3M".

Data Types: double

CTELength — Length of CTE

2 (default) | integer in the range [2, 20]

Length of the CTE, specified as an integer in the range [2, 20]. This value specifies the length of the CTE in 8 microsecond units.

Dependencies

To enable this property, apply these configurations.

- Set the Mode property to `"LE1M"` or `"LE2M"`.
- Set the PacketType property to `"ConnectionCTE"`.

Data Types: `double`

CTEType — Type of CTE

`[0; 0]` (default) | `[0; 1]` | `[1; 0]`

Type of CTE, specified as `[0; 0]`, `[0; 1]`, or `[1; 0]`.

Dependencies

To enable this property, apply these configurations.

- Set the Mode property to `"LE1M"` or `"LE2M"`.
- Set the PacketType property to `"ConnectionCTE"`.

Data Types: `double` | `logical`

Examples

Create Bluetooth BR/EDR or LE Test Waveform Configuration Object

Create a default Bluetooth BR/EDR or LE test waveform configuration object.

```
cfgTestWaveform = bluetoothTestWaveformConfig
```

```
cfgTestWaveform =
  bluetoothTestWaveformConfig with properties:

                Mode: 'LE1M'
       PayloadLength: 255
          PacketType: 'Disabled'
         PayloadType: 0
    SamplesPerSymbol: 8
```

Create a Bluetooth BR/EDR or LE test waveform configuration object, specifying the PHY transmission mode as EDR2M. Set the packet type to 2-DH1 and disable the data whiten status.

```
cfgEDR2MTestWaveform = bluetoothTestWaveformConfig(Mode="EDR2M", ...
    PacketType="2-DH1", ...
    WhitenStatus="Off")
```

```
cfgEDR2MTestWaveform =
  bluetoothTestWaveformConfig with properties:

                  Mode: 'EDR2M'
          WhitenStatus: 'Off'
       ModulationIndex: 0.3200
            PacketType: '2-DH1'
           PayloadType: 0
```

```
      SamplesPerSymbol: 8

   Read-only properties:
    FixedPayloadLength: 54
```

Create a default Bluetooth BR/EDR or LE test waveform configuration object. Set the PHY transmission mode to LE2M, packet type to connection-oriented CTE, and CTE length to 3.

```
cfgLE2MTestWaveform = bluetoothTestWaveformConfig;
cfgLE2MTestWaveform.Mode = "LE2M";
cfgLE2MTestWaveform.PacketType = "ConnectionCTE";
cfgLE2MTestWaveform.CTELength = 3
```

```
cfgLE2MTestWaveform =
  bluetoothTestWaveformConfig with properties:

                Mode: 'LE2M'
       PayloadLength: 255
          PacketType: 'ConnectionCTE'
         PayloadType: 0
    SamplesPerSymbol: 8
           CTELength: 3
             CTEType: [2x1 double]
```

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 27, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bluetoothTestWaveform` | `bleWaveformGenerator` | `bluetoothWaveformGenerator` | `bluetoothIdealReceiver` | `bleIdealReceiver` | `bleCTEIQSample`

Objects

`bluetoothRFPHYTestConfig` | `bluetoothWaveformConfig` | `bluetoothPhyConfig`

Topics
“Bluetooth LE Output Power and In-Band Emissions Tests”
“Bluetooth LE Blocking, Intermodulation and Carrier-to-Interference Performance Tests”
“Bluetooth LE Modulation Characteristics, Carrier Frequency Offset and Drift Tests”
“Bluetooth EDR RF-PHY Transmitter Tests for Modulation Accuracy and Carrier Frequency Stability”
“Bluetooth BR RF-PHY Transmitter Tests for Modulation Characteristics, Carrier Frequency Offset, and Drift”
“Bluetooth LE IQ samples Coherency and Dynamic Range Tests”
“Bluetooth BR/EDR Power and Spectrum Tests”

bluetoothWaveformConfig

Bluetooth BR/EDR waveform generator configuration parameters

Description

The `bluetoothWaveformConfig` object parameterizes the `bluetoothWaveformGenerator` function to generate a Bluetooth basic rate/enhanced data rate (BR/EDR) waveform.

Creation

Syntax

```
cfgWaveform = bluetoothWaveformConfig
cfg = bluetoothWaveformConfig(Name,Value)
```

Description

`cfgWaveform = bluetoothWaveformConfig` creates a default Bluetooth BR/EDR waveform generator configuration object, `cfgWaveform`.

`cfg = bluetoothWaveformConfig(Name,Value)` sets properties on page 2-135 by using one or more name-value pairs. Enclose each property name in quotes. For example, `('Mode','EDR3M')` sets the physical layer (PHY) transmission mode to 3 Mbps.

Properties

Note For more information about Bluetooth BR/EDR waveform generator properties and their respective values, see Volume 2, Part B, Sections 6 and 7 of the Bluetooth Core Specification [2].

Mode — PHY transmission mode

`'BR'` (default) | `'EDR2M'` | `'EDR3M'`

PHY transmission mode, specified as `'BR'`, `'EDR2M'`, or `'EDR3M'`.

Data Types: `char` | `string`

PacketType — Packet type

`'FHS'` (default) | `'ID'` | `'NULL'` | `'POLL'` | `...`

Packet type, specified as one of these values:

- `'ID'`
- `'NULL'`
- `'POLL'`
- `'FHS'`

- 'DM1'
- 'HV1'
- 'HV2'
- 'HV3'
- 'DV'
- 'EV3'
- 'EV4'
- 'EV5'
- 'DH1'
- 'AUX1'
- 'DM3'
- 'DH3'
- 'DM5'
- 'DH5'
- '2-EV3'
- '3-EV3'
- '2-EV5'
- '3-EV5'
- '2-DH1'
- '3-DH1'
- '2-DH3'
- '3-DH3'
- '2-DH5'
- '3-DH5'

Data Types: char | string

PayloadLength — Payload length of packet

18 (default) | integer in the range [0, *X*]

Payload length of packet, specified as an integer in the range [0, *X*], where *X* depends on the "PacketType" on page 2-0 property. This value sets the number of bytes to be processed in a packet.

Dependencies

To enable this property, set the value of PacketType property to 'DM1', 'DH1', 'DM3', 'DH3', 'DM5', 'DH5', 'AUX1', or 'DV'.

Data Types: double

DeviceAddress — Bluetooth device address

'0123456789AB' (default) | 12-element character vector | string scalar denoting 6-octet hexadecimal value

Bluetooth device address, specified as a 12-element character vector or string scalar denoting a 6-octet hexadecimal value.

Data Types: `char` | `string`

`LogicalTransportAddress` — Logical transport address for packet
`[0; 0; 1]` (default) | 3-bit binary column vector

Logical transport address for the packet, specified as a 3-bit binary column vector. This property indicates the active destination Peripheral for a packet in the Central-to-Peripheral transmission slot.

Data Types: `double`

`HeaderControlBits` — Header control information
`[1; 1; 1]` (default) | 3-bit binary column vector

Header control information, specified as a 3-bit binary column vector. This property indicates the header control information consisting of these three fields:

Field	Size of Field	Field Information Indicates
FLOW	1 bit	Flow control information
ARQN	1 bit	Acknowledgment for successful reception of a cyclic redundancy check (CRC) packet
SEQN	1 bit	Sequencing scheme to order the packet stream

Data Types: `double`

`ModulationIndex` — Modulation Index
`0.32` (default) | scalar in the range [0.28, 0.35]

Modulation index, specified as a scalar in the range [0.28, 0.35]. This property is the modulation index that the object uses when performing Gaussian frequency shift keying (GFSK) modulation or demodulation.

Data Types: `double`

`SamplesPerSymbol` — Samples per symbol
`8` (default) | positive integer

Samples per symbol, specified as a positive integer. This value is used for GFSK modulation and demodulation.

Data Types: `double`

`WhitenStatus` — Data whiten status
`'On'` (default) | `'Off'`

Data whiten status, specified as `'On'` or `'Off'`. Set this value to `'On'` for the object to perform whitening on the header and payload bits.

Data Types: `char` | `string`

`WhitenInitialization` — Whiten initialization
`[1; 1; 1; 1; 1; 1; 1]` (default) | 7-bit binary column vector

Whiten initialization, specified as a 7-bit binary column vector.

Dependencies

To enable this property, set the “WhitenStatus” on page 2-0 property to `'On'`.

Data Types: `double`

`LLID` — Logical link identifier

`[1; 1]` (default) | 2-bit binary column vector

Logical link identifier, specified as a 2-bit binary column vector.

Dependencies

To enable this property, set the “PacketType” on page 2-0 property to `'DM1'`, `'DH1'`, `'DM3'`, `'DH3'`, `'DM5'`, `'DH5'`, `'AUX1'`, and `'DV'`.

Data Types: `double`

`FlowIndicator` — Logical channel flow control indicator

`true` (default) | `false`

Logical channel flow control indicator, specified as `true` or `false`.

Dependencies

To enable this property, set the value of PacketType property to `'DM1'`, `'DH1'`, `'DM3'`, `'DH3'`, `'DM5'`, `'DH5'`, `'AUX1'`, and `'DV'`.

Data Types: `logical`

Note From R2022a, this object uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Object Functions

Specific to This Object

getPayloadLength	Return payload length for Bluetooth BR/EDR format configuration
getPhyConfigProperties	Return updated properties of Bluetooth BR/EDR PHY configuration object

Examples

Create Bluetooth BR/EDR Waveform Configuration Objects

Create three unique Bluetooth BR/EDR waveform configuration objects for synchronous connection oriented (SCO), connectionless peripheral broadcast (CPB), and asynchronous connectionless (ACL) logical transports.

Create a default Bluetooth BR/EDR waveform configuration object for an SCO logical transport. Set the packet type as HV1.

```
cfgWaveform = bluetoothWaveformConfig;
cfgWaveform.PacketType = 'HV1'
```

```
cfgWaveform =
  bluetoothWaveformConfig with properties:

                      Mode: 'BR'
                PacketType: 'HV1'
             DeviceAddress: '0123456789AB'
    LogicalTransportAddress: [3x1 double]
          HeaderControlBits: [3x1 double]
            ModulationIndex: 0.3200
           SamplesPerSymbol: 8
               WhitenStatus: 'On'
       WhitenInitialization: [7x1 double]
```

Create a second Bluetooth BR/EDR waveform configuration object for a CSB logical transport, specifying the packet type as DH1. Disable the whiten status.

```
cfgWaveform = bluetoothWaveformConfig('PacketType','DH1','WhitenStatus','Off')
```

```
cfgWaveform =
  bluetoothWaveformConfig with properties:

                      Mode: 'BR'
                PacketType: 'DH1'
             PayloadLength: 18
             DeviceAddress: '0123456789AB'
    LogicalTransportAddress: [3x1 double]
          HeaderControlBits: [3x1 double]
            ModulationIndex: 0.3200
           SamplesPerSymbol: 8
               WhitenStatus: 'Off'
                       LLID: [2x1 double]
             FlowIndicator: 1
```

Create a third Bluetooth BR/EDR waveform configuration object for an ACL logical transport with enhanced data rate mode. Set the value of packet type to DH3, and payload length to 184 bytes.

```
cfgWaveform = bluetoothWaveformConfig;
cfgWaveform.Mode = 'EDR2M';
cfgWaveform.PacketType = 'DH3';
cfgWaveform.PayloadLength = 184  % In bytes
```

```
cfgWaveform =
  bluetoothWaveformConfig with properties:

                      Mode: 'EDR2M'
                PacketType: 'DH3'
             PayloadLength: 184
             DeviceAddress: '0123456789AB'
    LogicalTransportAddress: [3x1 double]
          HeaderControlBits: [3x1 double]
            ModulationIndex: 0.3200
           SamplesPerSymbol: 8
               WhitenStatus: 'On'
       WhitenInitialization: [7x1 double]
                       LLID: [2x1 double]
             FlowIndicator: 1
```

Version History

Introduced in R2020a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

Usage notes and limitations:

Properties must be specified as coder.Constant().

See Also

Functions

`bluetoothWaveformGenerator` | `bluetoothIdealReceiver` | `bleWaveformGenerator` | `bleIdealReceiver`

Objects

`bluetoothPhyConfig`

Topics

“Bluetooth Packet Structure”
“Configure Bluetooth BR/EDR Channel with WLAN Interference and Pass the Waveform Through Channel”
“Bluetooth BR/EDR Waveform Generation and Transmission Using SDR”
“Bluetooth BR/EDR Waveform Reception Using SDR”

networkTrafficOnOff

On-Off application traffic pattern generator

Description

The `networkTrafficOnOff` object specifies the configuration parameters to generate an On-Off application traffic pattern.

You can use the On-Off application traffic pattern in Bluetooth, WLAN (requires WLAN Toolbox™) and 5G (requires 5G Toolbox™) system-level simulations to accurately model the real-world data traffic.

Creation

Syntax

```
cfgOnOff = networkTrafficOnOff
cfgOnOff = networkTrafficOnOff(Name,Value)
```

Description

`cfgOnOff = networkTrafficOnOff` creates a default On-Off application traffic pattern object.

`cfgOnOff = networkTrafficOnOff(Name,Value)` sets properties on page 2-141 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'GeneratePacket',true` generates an application packet.

Properties

`OnTime` — On state duration
[ ] (default) | positive scalar

On state duration, specified as a positive scalar. Units are in seconds. To specify a customized value for the On time, specify this property. If you do not specify this property, the object uses the exponential distribution to calculate the On time.

Data Types: `double`

`OffTime` — Off state duration
[ ] (default) | nonnegative scalar

Off state duration, specified as a nonnegative scalar. Units are in seconds. To specify a customized value for the Off time, specify this property. If you do not specify this property, the object uses the exponential distribution to calculate the Off time.

Data Types: `double`

`OnExponentialMean` — Exponential distribution mean value to calculate On state duration
1 (default) | positive scalar

Exponential distribution mean value to calculate the On state duration, specified as a positive scalar. Units are in seconds.

Dependencies

To enable this property, set the `OnTime` property to `[ ]`.

Data Types: `double`

`OffExponentialMean` — Exponential distribution mean value to calculate Off state duration

`2` (default) | nonnegative scalar

Exponential distribution mean value to calculate the Off state duration, specified as a nonnegative scalar. Units are in seconds.

Dependencies

To enable this property, set the `OffTime` property to `[ ]`.

Data Types: `double`

`DataRate` — Packet generation rate during On state

`5` (default) | positive scalar

Packet generation rate during the On state, specified as a positive scalar. Units are in Kbps. If the value of this property is low and the `PacketSize` is large, the object might not generate packets in the On state.

Data Types: `double`

`PacketSize` — Length of packet to be generated

`1500` (default) | positive scalar

Length of the packet to be generated, specified as a positive scalar. Units are in bytes. If the value of this property is greater than the `DataRate` property value, the object accumulates the data across multiple On times to generate a packet.

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate application packet

`false` or `0` (default) | `true` or `1`

Flag to indicate whether the object generates an application packet, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ApplicationData` — Application data to be added in packet

1500-by-1 vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the packet, specified as a column vector of integers in the range [0, 255]. If the size of the application data is greater than the `PacketSize` property value, the object truncates the application data. If the size of the application data is less than the `PacketSize` property value, the object appends zeros.

Dependencies

To enable this property, set the `GeneratePacket` property to `1` (`true`).

Data Types: `double`

Object Functions

Specific to This Object

generate Generate next On-Off application traffic packet

Examples

Generate On-Off Application Traffic Pattern

Create a default On-Off application traffic pattern object.

```
cfgOnOff = networkTrafficOnOff;
```

Generate an On-Off application traffic pattern.

```
[dt,packetSize] = generate(cfgOnOff);
```

Generate On-Off Application Traffic Pattern Using On State Mean Value

Create an On-Off application traffic pattern object, specifying the exponentially distributed mean value of the On state.

```
cfgOnOff = networkTrafficOnOff('OnExponentialMean',5);
```

Generate an On-Off application traffic pattern.

```
[dt,packetSize] = generate(cfgOnOff);
```

Generate On-Off Application Traffic Pattern and Data Packet

Create an On-Off application traffic pattern object to generate an On-Off data packet.

```
cfgOnOff = networkTrafficOnOff('GeneratePacket',true);
```

Generate an On-Off application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgOnOff);
```

Generate On-Off Application Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default On-Off application traffic pattern object.

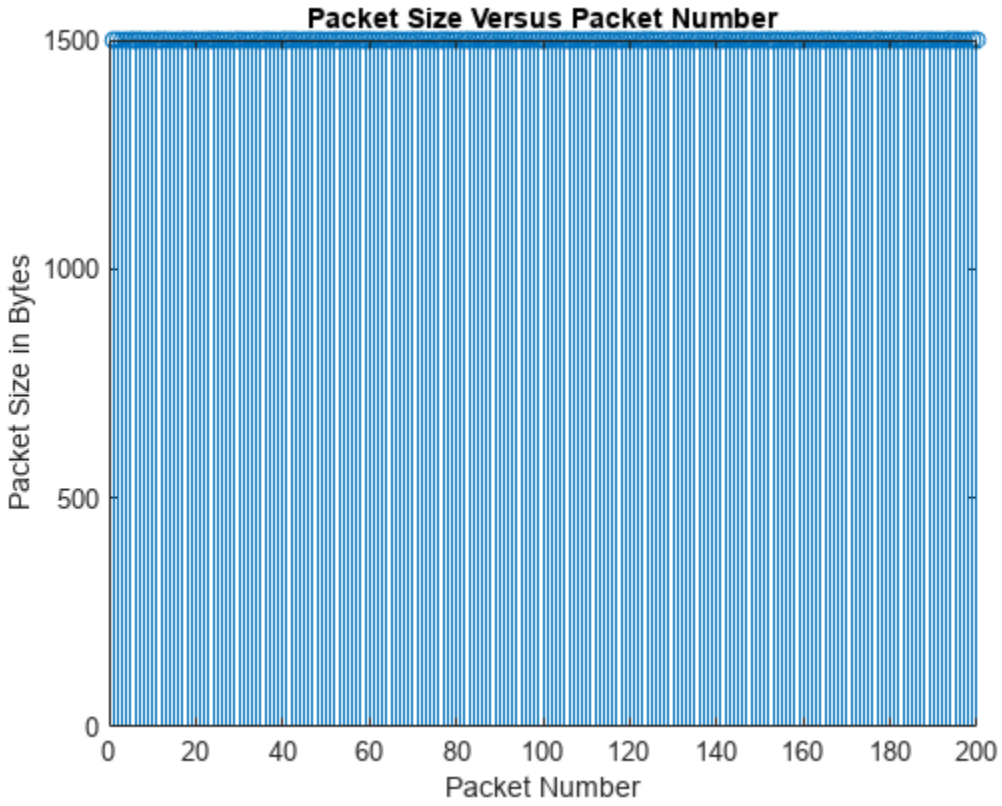
```
cfgOnOff = networkTrafficOnOff;
```

Generate an On-Off application traffic pattern with 200 On-Off packets.

```
for packetCount = 1:200
    [dt(packetCount),packetSize(packetCount)] = generate(cfgOnOff);
end
```

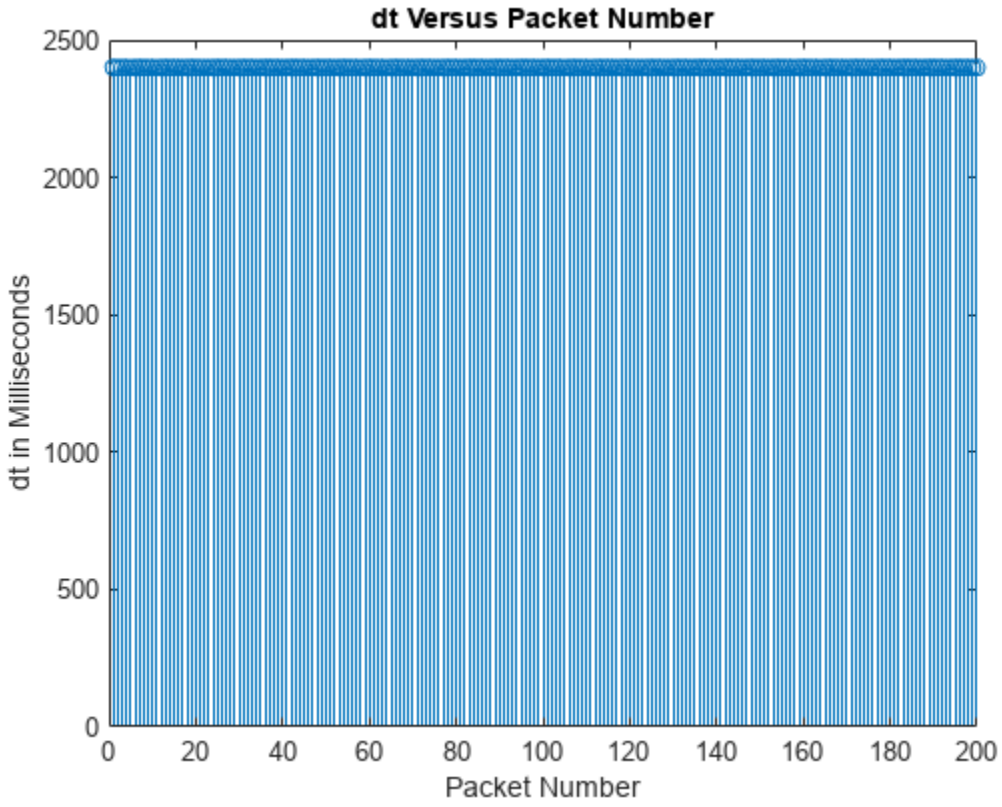
Visualize the On-Off packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the On-Off packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation

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

pcapngWriter

PCAPNG file writer of protocol packets

Description

The `pcapngWriter` object writes generated and recovered protocol packets to a packet capture next generation (PCAPNG) file (`.pcapng`).

You can write these packet types to a PCAPNG file:

- Generated and recovered Bluetooth low energy (LE) link layer (LL) packets (requires Bluetooth Toolbox)
- Generated and recovered 5G NR protocol packets (requires 5G Toolbox)
- Generated and recovered WLAN protocol packets (requires WLAN Toolbox)

Creation

Syntax

```
pcapngObj = pcapngWriter
pcapngObj = pcapngWriter(Name,Value)
```

Description

`pcapngObj = pcapngWriter` creates a default PCAPNG file writer object.

`pcapngObj = pcapngWriter(Name,Value)` sets properties on page 2-146 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'ByteOrder','big-endian'` specifies the byte order as big-endian.

Properties

Note The `pcapngWriter` object does not overwrite the existing PCAPNG file. Each time when you create this object, specify a unique PCAPNG file name.

`FileName` — Name of the PCAPNG file

`'capture'` (default) | character row vector | string scalar

Name of the PCAPNG file, specified as a character row vector or a string scalar.

Data Types: `char` | `string`

`ByteOrder` — Byte order

`'little-endian'` (default) | `'big-endian'`

Byte order, specified as `'little-endian'` or `'big-endian'`.

Data Types: `char` | `string`

`FileComment` — Comment for PCAPNG file
`''` (default) | character vector | string scalar

Comment for the PCAPNG file, specified as a character vector or a string scalar.

Data Types: `char` | `string`

Object Functions

Specific to This Object

write	Write protocol packet data to PCAP or PCAPNG file
writeCustomBlock	Write custom block to PCAPNG file
writeInterfaceDescriptionBlock	Write interface description block to PCAPNG file

Examples

Write Bluetooth LE Link Layer Packet to PCAPNG File

Create a PCAPNG file writer object, specifying the name of the PCAPNG file. Specify the Bluetooth LE link type.

```
pcapngObj = pcapngWriter('FileName','BLELLCapture');
```

Write an interface description block for Bluetooth LE.

```
interfaceName = 'Bluetooth LE interface';
bleLinkType = 251;
interfaceId = writeInterfaceDescriptionBlock(pcapngObj,bleLinkType, ...
    interfaceName);
```

Specify a Bluetooth LE LL packet.

```
llpacket = '42BC13E206120E00050014010A001F00400017001700000007D47C0';
```

Write the Bluetooth LE LL packet to the PCAPNG format file.

```
timestamp = 0;                                  % Packet arrival time in POSIX® microseconds elaps
packetComment = 'This is a Bluetooth LE packet';
write(pcapngObj,llpacket,timestamp,interfaceId,'PacketComment', ...
    packetComment);
```

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`pcapWriter`

pcapWriter

PCAP file writer of protocol packets

Description

The `pcapWriter` object writes generated and recovered protocol packets to a packet capture (PCAP) file (`.pcap`).

You can write these packet types to a PCAP file:

- Generated and recovered Bluetooth low energy (LE) link layer (LL) packets (requires Bluetooth Toolbox)
- Generated and recovered 5G NR protocol packets (requires 5G Toolbox)
- Generated and recovered WLAN protocol packets (requires WLAN Toolbox)

Creation

Syntax

```
pcapObj = pcapWriter
pcapObj = pcapWriter(Name,Value)
```

Description

`pcapObj = pcapWriter` creates a default PCAP file writer object.

`pcapObj = pcapWriter(Name,Value)` sets properties on page 2-149 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'ByteOrder','big-endian'` specifies the byte order as big-endian.

Properties

Note The `pcapWriter` object does not overwrite the existing PCAP file. Each time when you call this object, specify a unique PCAP file name.

`FileName` — Name of the PCAP file

`'capture'` (default) | character row vector | string scalar

Name of the PCAP file, specified as a character row vector or a string scalar.

Data Types: `char` | `string`

`ByteOrder` — Byte order

`'little-endian'` (default) | `'big-endian'`

Byte order, specified as `'little-endian'` or `'big-endian'`.

Data Types: `char` | `string`

Object Functions

Specific to This Object

write	Write protocol packet data to PCAP or PCAPNG file
writeGlobalHeader	Write global header to PCAP file

Examples

Write Bluetooth LE Link Layer Packet to PCAP File

Create a PCAP file writer object, specifying the name of the PCAP file. Specify the Bluetooth LE link type.

```
pcapObj = pcapWriter('FileName','writeBluetoothLE');
bleLinkType = 251;
```

Write a global header to the PCAP file.

```
writeGlobalHeader(pcapObj,bleLinkType);
```

Specify a Bluetooth LE LL packet.

```
llpacket = '42BC13E206120E00050014010A001F00400017001700000007D47C0';
```

Write the Bluetooth LE LL packet to the PCAP file.

```
timestamp = 129100;                % Packet arrival time in POSIX® microseconds elapsed since 1/1
write(pcapObj,llpacket,timestamp);
```

Version History

Introduced in R2020b

References

[1] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[2] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects
`pcapngWriter`

bleChannelSelection

Bluetooth LE channel index selection

Description

The `bleChannelSelection` System object™ selects a Bluetooth low energy (LE) channel index based on the algorithm specified by the "Algorithm" on page 2-0 property. This System object enables you to select the Bluetooth LE channel index for connection, periodic advertising, and isochronous events.

To select a Bluetooth LE channel index:

1. Create the `bleChannelSelection` 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

```
csa = bleChannelSelection
csa = bleChannelSelection(Name,Value)
```

Description

`csa = bleChannelSelection` creates a default Bluetooth LE channel selection System object, `csa`.

`csa = bleChannelSelection(Name,Value)` sets "Properties" on page 2-152 using one or more name-value pairs. Enclose each property name in quotes. For example, `('Algorithm',2)` specifies a Bluetooth LE channel index based on Algorithm 2.

Properties

Note For more information about Bluetooth LE channel selection properties, see Volume 6, Part B, Section 4.5.8 of the Bluetooth Core Specification [2].

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.

`Algorithm` — Type of Bluetooth LE channel selection algorithm

`1` (default) | `2`

Type of Bluetooth LE channel selection algorithm, specified as `1` or `2` representing Algorithm 1 or Algorithm 2, respectively. Algorithm 1 selects the channel index for connection events. Algorithm 2 selects the channel index for connection events, periodic advertising events, and LE isochronous events.

Data Types: `double`

`HopIncrement` — Hop increment count

`5` (default) | integer in the range [5, 16]

Hop increment count, specified as an integer in the range [5, 16]. This property specifies the hop increment count to hop between data channels. If you set the "Algorithm" on page 2-0 property to `1`, the System object uses this property as an input argument.

Data Types: `double`

`AccessAddress` — Unique connection address

`'8E89BED6'` (default) | 8-element character vector | string scalar denoting a 4-octet hexadecimal value

Unique connection address, specified as an 8-element character vector or a string scalar denoting a 4-octet hexadecimal value. This value specifies a unique 32-bit address for the link layer (LL) connection between two devices. If you set the "Algorithm" on page 2-0 property to `2`, the System object uses this property as an input argument.

Data Types: `char` | `string`

`UsedChannels` — List of used (good) data channels

row vector of integers in the range [0, 36]

List of used (good) data channels, specified as a vector of integers in the range [0, 36]. The vector length must be greater than 1. At least two channels must be set as used (good) channels. This property specifies the set of good channels classified by the Central.

Data Types: `double`

`SubeventChannelSelection` — Flag to enable isochronous event channel selection

`0` or `false` (default) | `1` or `true`

Flag to enable or disable isochronous event channel selection, specified as `0` (`false`) or `1` (`true`). A value of `1` (`true`) value selects a channel for isochronous events by specifying the "Algorithm" on page 2-0 property as `2`.

Data Types: `logical`

`IsochronousEventCounter` — Isochronous event counter

`0` (default) | integer in the range [0, 65,535]

Isochronous event counter, specified as an integer in the range [0, 65535]. This property specifies the counter for each isochronous event.

Dependencies

To enable this property, set the "SubeventChannelSelection" on page 2-0 property to `1` (`true`).

Data Types: `double`

`SubeventNumber` — Subevent counter
`1` (default) | integer in the range [1, 31]

Subevent counter, specified as an integer in the range [1, 31]. This property specifies the counter for each subevent.

Dependencies

To enable this property, set the "SubeventChannelSelection" on page 2-0 property to `1` (`true`).

Data Types: `double`

`ChannelIndex` — Channel index for current event
`0` (default) | integer in the range [0, 39]

This property is read-only.

Channel index for current event, specified as an integer in the range [0, 39]. This property specifies the channel index for connection event, periodic advertising event, and isochronous event.

Data Types: `double`

`EventCounter` — Counter for current event
`0` (default)

This property is read-only.

Counter for current event, specified as 0. This property specifies the event counter for connection event and periodic advertising event.

Data Types: `double`

Note From R2022a, this System object uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Usage

Syntax

```
channelIndex = csa
```

Description

`channelIndex = csa` selects a channel index based on the applicable "Properties" on page 2-152. During each call of this function, update the values of "IsochronousEventCounter" on page 2-0 and "SubeventNumber" on page 2-0 properties. During each call of this function, the System object must update the `SubeventNumber` property sequentially.

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 `bleChannelSelection`

clone	Create duplicate System object
isLocked	Determine if System object is in use

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Select Bluetooth LE Channel Indices Based on Type of Channel Selection Algorithm

Create a default Bluetooth LE channel selection System object.

```
csa = bleChannelSelection
```

```
csa =
  bleChannelSelection with properties:

       Algorithm: 1
    HopIncrement: 5
    UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ... ]
    ChannelIndex: 0
    EventCounter: 0
```

Set the hop increment count of 7. Then, specify a set of used (good) channels classified by the Central.

```
csa.HopIncrement = 7;
csa.UsedChannels = [0 2 24 6 10 14 26 30 34 36]
```

```
csa =
  bleChannelSelection with properties:

       Algorithm: 1
    HopIncrement: 7
    UsedChannels: [0 2 6 10 14 24 26 30 34 36]
    ChannelIndex: 0
    EventCounter: 0
```

Select a Bluetooth LE channel index by using the `csa` System object.

```
channelIndex = csa()
```

```
channelIndex = 30
```

Create another Bluetooth LE channel selection System object, specifying the type of channel selection algorithm as 2.

```
csa2 = bleChannelSelection('Algorithm',2);
```

Set a unique connection address. Then, specify a set of used (good) channels classified by the Central.

```
csa2.AccessAddress = '8E89BED6';
csa2.UsedChannels = [9 10 21 22 23 33 34 35 36]
```

```
csa2 =
  bleChannelSelection with properties:

                  Algorithm: 2
              AccessAddress: '8E89BED6'
    SubeventChannelSelection: false
                UsedChannels: [9 10 21 22 23 33 34 35 36]
                ChannelIndex: 0
                EventCounter: 0
```

Select a Bluetooth LE channel index by using the `csa2` System object.

```
channelIndex2 = csa2()
```

```
channelIndex2 = 35
```

Select Bluetooth LE Channel Indices for Isochronous Events

Create a default Bluetooth LE channel selection System object.

```
csa = bleChannelSelection
```

```
csa =
  bleChannelSelection with properties:

       Algorithm: 1
    HopIncrement: 5
    UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ... ]
    ChannelIndex: 0
    EventCounter: 0
```

Set the Bluetooth LE channel selection algorithm to 2. This algorithm supports channel selection for connection events, periodic advertising events, and LE isochronous events. Enable subevent channel selection.

```
csa.Algorithm = 2;
csa.SubeventChannelSelection = true;
```

Specify a unique connection address and a set of used (good) channels classified by the Central.

```
csa.AccessAddress = '8E89BED6';
csa.UsedChannels  = [9 10 21 22 23 33 34 35 36];
```

In the first isochronous event, select a channel index for the first subevent.

```
csa.IsochronousEventCounter = 0;
csa.SubeventNumber = 1;
channelIndex = csa()
```

```
channelIndex = 35
```

In the first isochronous event, select a channel index for the second subevent.

```
csa.SubeventNumber = 2;
channelIndex = csa()
```

```
channelIndex = 10
```

Version History

Introduced in R2019b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects

`bluetoothFrequencyHop`

Topics

“Bluetooth Packet Structure”
“Bluetooth LE Channel Selection Algorithms”

bluetoothWhiten

Whiten or dewhiten input data bits

Description

The `bluetoothWhiten` System object whitens or dewhitens input data bits by using a linear feedback shift register (LFSR). The Bluetooth Core Specification v5.3 [2] specifies the generator polynomial $x^7 + x^4 + 1$ and initial values of LFSR.

To whiten or dewhiten input data bits:

1. Create the `bluetoothWhiten` 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?

Note For more information about Bluetooth data whitening and dewhitening, see Volume 2, Part B, Section 7.2 and Volume 6, Part B, Section 3.2 of the Bluetooth Core Specification v5.3 [2].

Creation

Syntax

```
whiten = bluetoothWhiten
whiten = bluetoothWhiten(Name=Value)
```

Description

`whiten = bluetoothWhiten` creates a default Bluetooth whitening or dewhitening System object.

`whiten = bluetoothWhiten(Name=Value)` sets properties on page 2-158 by using one or more optional name-value arguments. For example, `InitialConditionsSource="Input port"` sets the input port as the source of the initial conditions vector.

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.

`InitialConditionsSource` — Source of initial conditions vector
`"Property"` (default) | `"Input port"`

Source of the initial conditions vector, specified as `"Property"` or `"Input port"`.

Data Types: `char` | `string`

`InitialConditions` — Initial conditions of LFSR
`[1;1;1;1;1;1;1]` (default) | seven-element column vector of binary values

Initial conditions of the LFSR, specified as a seven-element column vector of binary values. For the System object to generate an output different from the input data, at least one element of this property must be nonzero.

Data Types: `double`

Usage

Syntax

```
y = whiten(x)
y = whiten(x,initCondition)
```

Description

`y = whiten(x)` whitens or dewhitens the input data bits, `x`, at the transmitter or receiver, respectively, and returns the result, `y`.

`y = whiten(x,initCondition)` whitens or dewhitens the input data by using the specified initial values of the LFSR, `initCondition`.

Input Arguments

`x` — Input data bits
column vector of binary values

Input data, specified as a column vector of binary values.

Data Types: `double` | `logical`

`initCondition` — Initial conditions of LFSR
seven-element column vector of binary values

Initial conditions of LFSR, specified as a seven-element column vector of binary values. To specify this input,you must set the InitialConditionsSource property of `whiten` to `"Input port"`. For the System object to generate an output different from the input data, at least one element of this input must be nonzero.

Data Types: `double`

Output Arguments

`y` — Whitened or dewhitened output data bits
column vector of binary values

Whitened or dewhitened output data bits, returned as a column vector of binary values with the same length and data type as `x`.

Data Types: `double` | `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)
```

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

Whiten Random Data Based on Channel Index

Specify a channel index.

```
channelIndex = 4;
```

Generate initial conditions from the channel index.

```
initCond = [1; int2bit(channelIndex,6)];
```

Create a Bluetooth whiten System object, specifying the input port as the source of the initial conditions vector.

```
whiten = bluetoothWhiten(InitialConditionsSource="Input port");
```

Generate random input data.

```
data = randi([0 1],1024,1)
```

```
data = 1024×1

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1
     ⋮
```

Whiten the input data.

```
whitenedData = whiten(data,initCond)
```

```
whitenedData = 1024×1

     1
     1
     1
     1
     1
     1
     1
     1
     0
     1
      ⋮
```

Whiten and Dewhiten Random Data

Specify the initial conditions of a shift register.

```
initCond = [1 0 1 1 0 0 0]';
```

Create two System objects, one for whitening the input data, and the other for dewhitening the whitened data.

```
whiten = bluetoothWhiten(InitialConditions=initCond);
dewhiten = bluetoothWhiten(InitialConditions=initCond);
```

Generate random input data.

```
data = randi([0 1],800,1);
```

Whiten the input data.

```
whitenedData = whiten(data);
```

Perform dewhitening on the whitened data and, verify that the dewhitened output is same as the original, generated data.

```
deWhitenedData = dewhiten(whitenedData);
isequal(data,deWhitenedData)
```

```
ans = logical
   1
```

Version History

Introduced in R2022b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed March 22, 2022. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleWaveformGenerator` | `bleIdealReceiver` | `bluetoothWaveformGenerator` | `bluetoothIdealReceiver`

wirelessNetworkSimulator

Wireless network simulator

Description

The `wirelessNetworkSimulator` object simulates different wireless network scenarios with different types of wireless nodes. Use the Object Functions to add nodes to the simulator, interact with the nodes, schedule actions to perform during simulation, plug in custom channel models, and run simulations.

Creation

Syntax

```
networkSimulator = wirelessNetworkSimulator.init()
```

Description

`networkSimulator = wirelessNetworkSimulator.init()` creates a `wirelessNetworkSimulator` object with default property values. If the `wirelessNetworkSimulator` object `networkSimulator` already exists in the MATLAB® workspace, the function resets the object to its default state.

Properties

`CurrentTime` — Current simulation time

`0` (default) | non-negative scalar

This property is read-only.

Current simulation time, stored as a non-negative scalar. Units are in seconds.

Data Types: `double`

`ChannelFunction` — Channel model

`[ ]` (default) | function handle

This property is read-only.

Channel model, stored as a function handle. By default, the simulator uses the free space path loss (FSPL) model. To specify a custom channel model, use the `addChannelModel` function.

Data Types: `function_handle`

Object Functions

wirelessNetworkSimulator.getInstance	Get instance of wirelessNetworkSimulator object
addNodes	Add nodes to simulation

addChannelModel	Add custom channel or path loss model
scheduleAction	Schedule action to perform during simulation
cancelAction	Cancel scheduled action
run	Run simulation

Examples

Simulate Bluetooth BR Network with Default Channel Effects

Create a `wirelessNetworkSimulator` object. By default, the `wirelessNetworkSimulator` object applies free-space path loss model for the channel effects.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and other with the `"peripheral"` role. Specify the position of the Peripheral node in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
          GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Specify the simulation time in seconds.

```
simulationTime = 0.05;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve application, baseband, and physical layer (PHY) statistics corresponding to the Central and Peripheral nodes.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
        Name: "Node1"
          ID: 1
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]


peripheralStats = statistics(peripheralNode)

peripheralStats = struct with fields:
        Name: "Node2"
          ID: 2
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

Simulate Bluetooth BR Network without Channel Effects

Create a `wirelessNetworkSimulator` object.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and other with the `"peripheral"` role. Specify the position of the Peripheral node in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
    GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

By default, the `wirelessNetworkSimulator` object applies free-space path loss model for the channel effects. You can add custom channel effects by using the `addChannelModel` function.

However, to model the channel without any channel effects, specify a custom MATLAB™ function in which the input transmitted packets are returned at the output without any changes.

```
addChannelModel(networkSimulator,@removeChannelEffect);
```

Specify the simulation time in seconds.

```
simulationTime = 0.05;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

Retrieve application, baseband, and physical layer (PHY) statistics corresponding to the Central and Peripheral nodes.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
        Name: "Node1"
          ID: 1
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

```
peripheralStats = statistics(peripheralNode)
```

```
peripheralStats = struct with fields:
        Name: "Node2"
          ID: 2
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

```
function outputData = removeChannelEffect(~,txData)
outputData = txData;
end
```

Version History

Introduced in R2022b

See Also

`bluetoothLENode` | `bluetoothNode`

Topics

“Create, Configure, and Simulate Bluetooth BR Piconet”
“Bluetooth BR Data and Voice Communication with WLAN Signal Interference”
“Simulate Multiple Bluetooth BR Piconets with ACL Traffic”

Object Functions

addTrafficSource

Add data traffic source to Bluetooth LE node

Syntax

```
addTrafficSource(bluetoothLENodeObj,trafficSource)
addTrafficSource(bluetoothLENodeObj,trafficSource,
DestinationNode=destinationNode)
addTrafficSource(bluetoothLENodeObj,trafficSource,
DestinationAddress=destinationElementAddress,
SourceAddress=sourceElementAddress,varargin)
addTrafficSource(bluetoothLENodeObj,trafficSource,
DestinationAddress=destinationElementAddress,
SourceAddress=sourceElementAddress,TTL=ttl)
```

Description

`addTrafficSource(bluetoothLENodeObj,trafficSource)` adds a data traffic source, `trafficSource`, to the Bluetooth low energy (LE) node object `bluetoothLENodeObj`. The Role property of the `bluetoothLENode` object must be set to `"isochronous-broadcaster"`.

`addTrafficSource(bluetoothLENodeObj,trafficSource, DestinationNode=destinationNode)` adds a data traffic source to send data to the specified destination node `DestinationNode`. The Role property of `bluetoothLENode` object must be set to `"central"` or `"peripheral"`.

`addTrafficSource(bluetoothLENodeObj,trafficSource, DestinationAddress=destinationElementAddress, SourceAddress=sourceElementAddress,varargin)` adds a data traffic source to send data from the specified element of the source mesh node `SourceAddress` to the specified element of the destination mesh node `DestinationAddress`. The Role property of the `bluetoothLENode` object must be set to `"broadcaster-observer"`. `varargin` specifies one or more name-value arguments of the upper layer metadata.

`addTrafficSource(bluetoothLENodeObj,trafficSource, DestinationAddress=destinationElementAddress, SourceAddress=sourceElementAddress,TTL=ttl)` additionally specifies the time to live (TTL) for the data communication between the specified source and destination mesh nodes. The Role property of the `bluetoothLENode` object must be set to `"broadcaster-observer"`.

Examples

Create, Configure, and Simulate Bluetooth LE Network

This example shows you how to simulate a Bluetooth® low energy (LE) network by using Bluetooth® Toolbox.

Using this example, you can:

1. Create and configure a Bluetooth LE piconet with Central and Peripheral nodes.
2. Create and configure a link layer (LL) connection between Central and Peripheral nodes.
3. Add application traffic from the Central to Peripheral nodes.
4. Simulate Bluetooth LE network and retrieve the statistics of the Central and Peripheral nodes.

Create a wireless network simulator.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create a Bluetooth LE node, specifying the role as `"central"`. Specify the name and position of the node.

```
centralNode = bluetoothLENode("central");
centralNode.Name = "CentralNode";
centralNode.Position = [0 0 0];                 % In x-, y-, and z-coordinates in meters
```

Create a Bluetooth LE node, specifying the role as `"peripheral"`. Specify the name and position of the node.

```
peripheralNode = bluetoothLENode("peripheral");
peripheralNode.Name = "PeripheralNode";
peripheralNode.Position = [10 0 0]              % In x-, y-, and z-coordinates in meters
```

```
peripheralNode =
  bluetoothLENode with properties:

        TransmitterPower: 20
         TransmitterGain: 0
           ReceiverRange: 100
            ReceiverGain: 0
     ReceiverSensitivity: -100
             NoiseFigure: 0
    InterferenceFidelity: 0
                    Name: "PeripheralNode"
                Position: [10 0 0]

   Read-only properties:
                    Role: "peripheral"
        ConnectionConfig: [1x1 bluetoothLEConnectionConfig]
          TransmitBuffer: [1x1 struct]
                      ID: 2
```

Create a default Bluetooth LE configuration object to share the LL connection between the Central and Peripheral nodes.

```
cfgConnection = bluetoothLEConnectionConfig;
```

Specify the connection interval and connection offset. Throughout the simulation, the object establishes LL connection events for the duration of each connection interval. The connection offset is from the beginning of the connection interval.

```
cfgConnection.ConnectionInterval = 0.01; % In seconds
cfgConnection.ConnectionOffset = 0;      % In seconds
```

Specify the active communication period after the connection event is established between the Central and Peripheral nodes.

```
cfgConnection.ActivePeriod = 0.01 % In seconds
```

```
cfgConnection =
  bluetoothLEConnectionConfig with properties:

    ConnectionInterval: 0.0100
         AccessAddress: "5DA44270"
          UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
             Algorithm: 1
          HopIncrement: 5
     CRCInitialization: "012345"
    SupervisionTimeout: 1
               PHYMode: "LE1M"
         InstantOffset: 6
      ConnectionOffset: 0
          ActivePeriod: 0.0100
```

Configure the connection between Central and Peripheral nodes by using the `configureConnection` object function of the `bluetoothLEConnectionConfig` object.

```
configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Specify the data rate in kb/s and the packet size in bytes. Enable packet generation to generate an application packet with a payload.

```
traffic = networkTrafficOnOff(DataRate=100, ...
                              PacketSize=10, ...
                              GeneratePacket=true);
```

Add application traffic from the Central to the Peripheral node by using the `addTrafficSource` object function.

```
addTrafficSource(centralNode,traffic,"DestinationNode",peripheralNode.Name);
```

Create a Bluetooth LE network consisting of a Central and a Peripheral node.

```
nodes = {centralNode peripheralNode};
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,nodes)
```

Set the simulation time in seconds and run the simulation.

```
simulationTime = 0.5;
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve application, link layer (LL), and physical layer (PHY) statistics corresponding to the broadcaster and receiver nodes. For more information about the statistics, see “Bluetooth LE Node Statistics”.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
    Name: "CentralNode"
```

```
      ID: 1
     App: [1x1 struct]
      LL: [1x1 struct]
     PHY: [1x1 struct]


peripheralStats = statistics(peripheralNode)

peripheralStats = struct with fields:
    Name: "PeripheralNode"
      ID: 2
     App: [1x1 struct]
      LL: [1x1 struct]
     PHY: [1x1 struct]
```

Input Arguments

`bluetoothLENodeObj` — Bluetooth LE node object

`bluetoothLENode` object

Bluetooth LE node object, specified as a `bluetoothLENode` object.

`trafficSource` — On-Off application traffic pattern object

`networkTrafficOnOff` object

On-Off application traffic pattern object, specified as a `networkTrafficOnOff` object.

`DestinationNode` — Name of the destination node

character vector | string scalar | `bluetoothLENode` object

Name of the destination node, specified as a character vector, a string scalar, or a `bluetoothLENode` object. If you set the Role property of the `bluetoothLENodeObj` object to `"central"` or `"peripheral"`, this input is mandatory. This value specifies the Name property of the specified Bluetooth LE node object `bluetoothLENodeObj` for sending data to the specified destination node.

Data Types: `char` | `string`

`DestinationAddress` — Destination address

4-element character vector | string scalar denoting a 2-octet hexadecimal address

Destination address, specified as a 4-element character vector or string scalar denoting a 2-octet hexadecimal address. If you set the Role property of the `bluetoothLENodeObj` object to `"broadcaster-observer"`, this input is mandatory. This value specifies the destination address of the Bluetooth mesh node.

Data Types: `char` | `string`

`SourceAddress` — Source address

4-element character vector | string scalar denoting a 2-octet hexadecimal address

Source address, specified as a 4-element character vector or string scalar denoting a 2-octet hexadecimal address. If you set the Role property of the `bluetoothLENodeObj` object to `"broadcaster-observer"`, this input is mandatory. This value specifies the element address of the Bluetooth mesh source node. The element address specifies the ElementAddress property of the specified Bluetooth mesh profile configuration object `bluetoothMeshProfileConfig`.

Data Types: `char` | `string`

TTL — Time to live for message
127 (default) | integer in the range [0, 127], excluding 1

Time to live for the message, specified as an integer in the range [0, 127], excluding 1. If you set the Role property of the `bluetoothLENode` object to `"broadcaster-observer"`, this value is optional. If the value is not set, the function uses the default TTL property value `127` of the `bluetoothMeshProfileConfig` object. This value specifies the maximum number of hops that the transmitted message can traverse between the source and destination elements. For more information about the TTL value, see Bluetooth mesh profile v1.0.1, section 4.2.7.

Data Types: `double`

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 12, 2021. https://www.bluetooth.com/.

[2] Bluetooth Core Specifications Working Group. "Bluetooth Core Specification" v5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothLENode`

Functions
`channelInvokeDecision` | `pushChannelData` | `runNode` | `statistics` | `updateChannelList`

write

Write Bluetooth LE LL protocol packet data to PCAP or PCAPNG file

Syntax

```
write(pcapObj,packet,timestamp)
write(pcapObj,packet,timestamp,Name,Value)
```

Description

`write(pcapObj,packet,timestamp)` writes Bluetooth low energy (LE) link layer (LL) protocol packet data to the packet capture (PCAP) or packet capture next generation (PCAPNG) file specified in the Bluetooth LE PCAP file writer object, `pcapObj`. Input `packet` specifies the Bluetooth LE LL protocol packet, and input `timestamp` specifies the packet arrival time.

`write(pcapObj,packet,timestamp,Name,Value)` specifies options using one or more name-value pair arguments. For example, `'PacketFormat','bits'` sets the format of the Bluetooth LE LL protocol packets to bits.

Examples

Write Bluetooth LE LL Packet to PCAP File

Create a default Bluetooth LE PCAP file writer object.

```
pcapObj = blePCAPWriter;
```

Generate a Bluetooth LE LL packet.

```
cfgLLData = bleLLDataChannelPDUConfig('LLID', ...
    'Data (start fragment/complete)');
payload = '0E00050014010A001F004000170017000000';
llDataPDU = bleLLDataChannelPDU(cfgLLData,payload);
connAccessAddress = de2bi(hex2dec('E213BC42'),32)';
llpacket = [connAccessAddress;llDataPDU];
```

Write the Bluetooth LE LL packet to the PCAP file.

```
timestamp = 0;                                          % Packet arrival time in POSIX® microsecon
write(pcapObj,llpacket,timestamp,'PacketFormat','bits');
```

Use Bluetooth LE PCAP Writer to Write LL Packet to PCAPNG File

Create a Bluetooth LE PCAPNG file writer object, specifying the name and extension of the PCAPNG file.

```
pcapObj = blePCAPWriter('FileName','sampleBLELL', ...
    'FileExtension','pcapng');
```

Generate a Bluetooth LE LL packet.

```
cfgLLData = bleLLDataChannelPDUConfig('LLID', ...
    'Data (start fragment/complete)');
payload = '0E00050014010A001F004000170017000000';
llDataPDU = bleLLDataChannelPDU(cfgLLData,payload);
connAccessAddress = de2bi(hex2dec('E213BC42'),32)';
llpacket = [connAccessAddress; llDataPDU];
```

Write the Bluetooth LE LL packet to the PCAPNG file.

```
timestamp = 12800000;                                % Packet arrival time in POSIX® microseconds
write(pcapObj,llpacket,timestamp,'PacketFormat','bits');
```

Input Arguments

Note The `blePCAPWriter` object does not overwrite the existing PCAP or PCAPNG file. Each time when you create this object, specify a unique PCAP or PCAPNG file name.

`pcapObj` — Bluetooth LE PCAP file writer object

`blePCAPWriter` object

Bluetooth LE PCAP file writer object, specified as a `blePCAPWriter` object.

`packet` — Bluetooth LE LL protocol packet

binary-valued vector | character vector | string scalar | numeric vector | *n*-by-2 character array

Bluetooth LE LL protocol packet, specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format.

Data Types: `char` | `string` | `double`

`timestamp` — Packet arrival time

nonnegative integer

Packet arrival time in POSIX® microseconds elapsed since 1/1/1970, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'PacketFormat','bits'` sets the format of the Bluetooth LE LL protocol packets to bits.

`PhyHeader` — Bluetooth LE LL protocol packet metadata
binary-valued vector | character vector | string scalar | numeric vector | *n*-by-2 character array

Bluetooth LE LL protocol packet metadata, specified as the comma-separated pair consisting of `PhyHeader` and one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format.

Data Types: `char` | `string` | `double`

`PacketComment` — Comment for the Bluetooth LE LL protocol packet
`''` (default) | character vector | string scalar

Comment for the Bluetooth LE LL protocol packet, specified as the comma-separated pair consisting of `PacketComment` and a character vector or a string scalar.

Data Types: `char` | `string`

`PacketFormat` — Format of the Bluetooth LE LL protocol packet
`'octets'` (default) | `'bits'`

Format of the Bluetooth LE LL protocol packet, specified as the comma-separated pair consisting of `PacketFormat` and `'octets'` or `'bits'`. If this value is specified as `'octets'`, `packet` is specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format.

Data Types: `char` | `string` | `double`

`PhyHeaderFormat` — Format of the PHY header
`'octets'` (default) | `'bits'`

Format of the physical layer (PHY) header, specified as the comma-separated pair consisting of `PhyHeaderFormat` and `'octets'` or `'bits'`. If you specify this value as `'octets'`, the `PhyHeader` argument can be specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.

- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format

Data Types: `char` | `string` | `double`

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`blePCAPWriter`

addTrafficSource

Add data traffic source to Bluetooth BR node on ACL connection

Syntax

```
addTrafficSource(bluetoothBRNodeObj,trafficSource,
DestinationNode=connectedNode)
```

Description

`addTrafficSource(bluetoothBRNodeObj,trafficSource, DestinationNode=connectedNode)` adds a data traffic source object, `trafficSource`, to the specified Bluetooth basic rate (BR) node for sending data to the specified connected destination node, `connectedNode`.

Examples

Create, Configure, and Simulate Bluetooth BR Nodes

Initialize the wireless network simulator.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and the other with the `"peripheral"` role. Specify the position of the Peripheral node, in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between the Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure the connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode)
```

```
connection =
  bluetoothConnectionConfig with properties:

    CentralToPeripheralACLPacketType: "DH1"
    PeripheralToCentralACLPacketType: "DH1"
                       SCOPacketType: "None"
                  HoppingSequenceType: "Connection adaptive"
                         UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 ... ]
                         PollInterval: 40
                        InstantOffset: 240
                    TransmitterPower: 20
                   SupervisionTimeout: 32000
```

```
   Read-only properties:
                      CentralAddress: "D091BBE70001"
                    PrimaryLTAddress: 1
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
    GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Specify the simulation time, in seconds.

```
simulationTime = 0.3;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve the application, baseband, and physical layer (PHY) statistics corresponding to the Central and Peripheral nodes.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
        Name: "Node1"
          ID: 1
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

```
peripheralStats = statistics(peripheralNode)
```

```
peripheralStats = struct with fields:
        Name: "Node2"
          ID: 2
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

Input Arguments

`bluetoothBRNodeObj` — Bluetooth BR node object

`bluetoothNode` object

Bluetooth BR node object, specified as a `bluetoothNode` object.

`trafficSource` — On-Off application traffic pattern object
`networkTrafficOnOff` object

On-Off application traffic pattern object, specified as a `networkTrafficOnOff` object. The object function uses this input to transport data traffic over the asynchronous connection-oriented logical (ACL) link of the connection.

`DestinationNode` — Name of the destination node
`bluetoothNode` object

Name of the destination node, specified as a `bluetoothNode` object.

Data Types: `char` | `string`

Version History

Introduced in R2022b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed May 22, 2022. https://www.bluetooth.com/.

[2] Bluetooth Core Specifications Working Group. "Bluetooth Core Specification" v5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothNode` | `bluetoothConnectionConfig`

Functions
`statistics` | `updateChannelList`

statistics

Get statistics of Bluetooth BR node

Syntax

```
nodeStatistics = statistics(bluetoothBRNodeObj)
```

Description

`nodeStatistics = statistics(bluetoothBRNodeObj)` returns the statistics of the Bluetooth basic rate (BR) node object `bluetoothBRNodeObj`.

Examples

Create, Configure, and Simulate Bluetooth BR Nodes

Initialize the wireless network simulator.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and the other with the `"peripheral"` role. Specify the position of the Peripheral node, in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between the Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure the connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode)
```

```
connection =
  bluetoothConnectionConfig with properties:

    CentralToPeripheralACLPacketType: "DH1"
    PeripheralToCentralACLPacketType: "DH1"
                         SCOPacketType: "None"
                  HoppingSequenceType: "Connection adaptive"
                         UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 ... ]
                         PollInterval: 40
                        InstantOffset: 240
                     TransmitterPower: 20
                   SupervisionTimeout: 32000

   Read-only properties:
                       CentralAddress: "D091BBE70001"
                    PrimaryLTAddress: 1
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
    GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Specify the simulation time, in seconds.

```
simulationTime = 0.3;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve the application, baseband, and physical layer (PHY) statistics corresponding to the Central and Peripheral nodes.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
        Name: "Node1"
          ID: 1
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

```
peripheralStats = statistics(peripheralNode)
```

```
peripheralStats = struct with fields:
        Name: "Node2"
          ID: 2
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

Input Arguments

`bluetoothBRNodeObj` — Bluetooth BR node object

`bluetoothNode` object

Bluetooth BR node object, specified as a `bluetoothNode` object.

Output Arguments

`nodeStatistics` — Statistics of Bluetooth BR node
structure

Statistics of the Bluetooth BR node, returned as a structure. For more information about this output, see “Bluetooth Node Statistics”.

Data Types: `struct`

Version History

Introduced in R2022b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed May 22, 2022. https://www.bluetooth.com/.

[2] Bluetooth Core Specifications Working Group. "Bluetooth Core Specification" v5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothNode` | `bluetoothConnectionConfig`

Functions
`addTrafficSource` | `updateChannelList`

Topics
“Bluetooth Node Statistics”

updateChannelList

Update channel list of Bluetooth BR node

Syntax

```
status = updateChannelList(bluetoothBRNodeObj,newUsedChannelsList,
DestinationNode=destinationNode)
```

Description

`status = updateChannelList(bluetoothBRNodeObj,newUsedChannelsList, DestinationNode=destinationNode)` updates the channel map for the physical link between the Bluetooth basic rate (BR) node, `bluetoothBRNodeObj`, and the specified destination node, `destinationNode`, by providing a new list of used channels, `newUsedChannelsList`, to the Bluetooth BR node, `bluetoothBRNodeObj`, object. The object function returns a status, `status`, indicating whether `bluetoothBRNodeObj` accepts the new channel list. To enable this syntax, set the Role property of the `bluetoothBRNodeObj` object to `"central"`.

Examples

Schedule Updated Channel Map for Bluetooth BR Node

Initialize the wireless network simulator.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and other with the `"peripheral"` role. Specify the position of the Peripheral node in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode)
```

```
connection =
  bluetoothConnectionConfig with properties:

    CentralToPeripheralACLPacketType: "DH1"
    PeripheralToCentralACLPacketType: "DH1"
                        SCOPacketType: "None"
                  HoppingSequenceType: "Connection adaptive"
                         UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 ... ]
                         PollInterval: 40
                        InstantOffset: 240
```

```
                  TransmitterPower: 20
                SupervisionTimeout: 32000

   Read-only properties:
                    CentralAddress: "D091BBE70001"
                  PrimaryLTAddress: 1
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
    GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Schedule channel list update at the Central node at 0.05 seconds to use channels 0 to 40.

```
scheduleAction(networkSimulator,@(varargin) updateChannelList(centralNode, ...
    0:40,DestinationNode=peripheralNode),[],0.05);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Specify the simulation time in seconds.

```
simulationTime = 0.3;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve application, baseband, and physical layer (PHY) statistics corresponding to the Central and Peripheral nodes.

```
centralStats = statistics(centralNode);
peripheralStats = statistics(peripheralNode);
```

Input Arguments

`bluetoothBRNodeObj` — Bluetooth BR node object

`bluetoothNode` object

Bluetooth BR node object, specified as a `bluetoothNode` object with the Role property set to `"central"`. To enable this input, perform these steps.

- Use the `configureConnection` object function to configure a connection between this node and the destination node.
- Set the HoppingSequenceType property of the `bluetoothConnectionConfig` object to `"Connection adaptive"`.

`newUsedChannelsList` — List of good channels
vector of integers in range [0, 78]

List of good channels, specified as a vector of integers in the range [0, 78]. This input specifies the new list of used channels with which to update the channel map for the specified Bluetooth BR node. To ensure that at least 20 channels are set as used (good) channels, specify this vector with unique values and a length greater than or equal to 20.

Data Types: `double`

`DestinationNode` — Name of destination node
`bluetoothNode` object

Name of the destination node, specified as a `bluetoothNode` object. The destination node specified by this input must be connected to the Bluetooth BR node specified by the `bluetoothBRNodeObj` input by using the `configureConnection` object function.

Output Arguments

`status` — Flag indicating whether baseband layer accepts new channel list
`0` or `false` | `1` or `true`

Flag indicating whether the baseband layer of the specified Bluetooth BR node accepts the new channel list, returned as a logical `0` (`false`) or `1` (`true`).

Data Types: `logical`

Version History

Introduced in R2022b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed May 22, 2022. https://www.bluetooth.com/.

[2] Bluetooth Core Specifications Working Group. "Bluetooth Core Specification" v5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothNode` | `bluetoothConnectionConfig`

Functions
`addTrafficSource` | `statistics`

channelInvokeDecision

(To be removed) Determine whether to apply channel to incoming signal

Note will be removed in a future release. To determine whether to apply channel to the incoming Bluetooth LE signal and simulate the Bluetooth LE node, use the `wirelessNetworkSimulator` object, instead.

Syntax

```
[flag,rxInfo] = channelInvokeDecision(bluetoothLENodeObj,packet)
```

Description

`[flag,rxInfo] = channelInvokeDecision(bluetoothLENodeObj,packet)` returns a flag, `flag`, indicating whether the Bluetooth low energy (LE) node specified by the `bluetoothLENodeObj` object wants to receive the packet `packet`. The function also returns the receiver information `rxInfo` required to apply the channel to the incoming packet.

Examples

Run Bluetooth LE Node

Create a Bluetooth LE node, specifying the role as `"central"`.

```
centralNode = bluetoothLENode("central");
```

Create a Bluetooth LE node, specifying the role as `"peripheral"`.

```
peripheralNode = bluetoothLENode("peripheral");
```

Create a default Bluetooth LE configuration object to establish a connection between the Central and Peripheral nodes.

```
cfgConnection = bluetoothLEConnectionConfig;
```

Configure the connection between the Central and Peripheral nodes.

```
configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Specify the data rate in kb/s and the packet size in bytes. Generate an application packet with a payload by enabling packet generation. Add application traffic from the Central to the Peripheral node by using the `addTrafficSource` object function.

```
traffic = networkTrafficOnOff(DataRate=50, ...
    PacketSize=20, ...
    GeneratePacket=true);
addTrafficSource(centralNode,traffic,"DestinationNode",peripheralNode.Name);
```

Create a Bluetooth LE network consisting of a Central and a Peripheral node.

```
nodes = {centralNode peripheralNode};
numNodes = length(nodes);
```

Set the current simulation time in seconds.

```
currentTime = 0;
```

Run the Bluetooth LE nodes at the current simulation time. The `runNode` object function returns the time instant at which the node runs again.

```
for index = 1:numNodes
    nextInvokeTime = runNode(nodes{index},currentTime);
end
```

If the transmit buffer has packet to be transmitted, the `channelInvokeDecision` object function performs these tasks.

- Determines whether the receiving node wants to receive the packet.
- Retrieves the receiver information required to apply the channel to the transmitted packet.

If the node wants to receive the packet, apply the channel to the transmitted packet and push the data from the channel to the reception buffer by using the `pushChannelData` object function.

```
for txIndex = 1:numNodes
    txNode = nodes{txIndex};
    if (txNode.TransmitBuffer.Type ~= 0)
        txPacket = txNode.TransmitBuffer;
        for rxIndex = 1:numNodes
            rxNode = nodes{rxIndex};
            [flag,rxInfo] = channelInvokeDecision(rxNode,txPacket);
            if flag
                % Apply a path loss to the transmitted data in this comment
                % line before pushing the data to the reception buffer
                pushChannelData(rxNode,txPacket);
            end
        end
    end
end
```

Input Arguments

`bluetoothLENodeObj` — Bluetooth LE node object

`bluetoothLENode` object

Bluetooth LE node object, specified as a `bluetoothLENode` object.

`packet` — Incoming packet

structure

Incoming packet, specified as a structure with these fields.

Structure Field	Structure Value	Description
`Type`	`0`, `1`, `2`, or `3`	Type of the incoming signal. Each value represents one of these packet types. • `0` — Invalid packet • `1` — WLAN packet • `2` — 5G packet • `3` — Bluetooth LE packet The default value is 0.
`TransmitterID`	nonnegative scalar integer	Transmitter node identifier
`TransmitterPosition`	numeric row vector of length three	Position of the transmitter in Cartesian x-, y-, and z-coordinates. The value of this field is in the form [X Y Z]. Units are in meters.
`TransmitterVelocity`	real-valued row vector of length three	Velocity of the transmitter in x-, y-, and z-directions. The value of this field is in the form [*vx vy vz*], where *v* is the velocity. Units are in meters per second.
`StartTime`	nonnegative integer	Packet transmission start time at the transmitter or packet arrival time at the receiver. Units are in seconds.
`Duration`	positive integer	Duration of the packet, in seconds
`Power`	scalar	Transmit power of the packet in dBm
`CenterFrequency`	scalar positive integer	Center frequency of the carrier in Hz
`Bandwidth`	scalar positive integer	Carrier bandwidth in Hz
`Abstraction`	`0` (`false`) or `1` (`true`)	Physical layer (PHY) abstraction type. A `1` (`true`) value represents the abstracted PHY. A `0` (`false`) value specifies the full PHY. The default value is `0` (`false`).
`SampleRate`	scalar positive integer	Sample rate of the packet in samples/second

Structure Field	Structure Value	Description
`Data`	*M* x *N* matrix	If you set the `Abstraction` field of the `packet` argument, this field specifies the in-phase and quadrature (IQ) samples of the packet as a *M*-by-*N* matrix, where *M* is the number of IQ samples and *N* is the number of Rx antennas. When the `Abstraction` field of the `packet` argument is set to `1` (`true`), this field has frame information.
`Metadata`	structure	Technology specific or abstraction specific information, specified as a structure. The default value is an empty structure.

Data Types: `struct`

Output Arguments

`flag` — Flag indicating whether to invoke channel

`0` or `false` | `1` or `true`

Flag indicating whether to invoke channel, returned as a `0` (`false`) or `1` (`true`).

Data Types: `logical`

`rxInfo` — Receiver information

structure

Receiver information, returned as a structure with these fields.

Structure Field	Structure Value	Description
`ID`	nonnegative scalar integer	Receiver node identifier
`Position`	numeric row vector of length three	Position of the receiver in Cartesian x-, y-, and z-coordinates. The value of this field is in the form [X Y Z]. Units are in meters.
`Velocity`	real-valued row vector of length three	Velocity of the receiver in x-, y-, and z-directions. The value of this field is in the form [*vx vy vz*], where *v* is the velocity. Units are in meters per second.

The object function uses this output to apply the channel to the incoming packet.

Data Types: `struct`

Version History

Introduced in R2022a

`channelInvokeDecision` object function will be removed
Not recommended starting in R2022b

`channelInvokeDecision` will be removed in a future release. To determine whether to apply channel to the incoming Bluetooth LE signal, use the `wirelessNetworkSimulator` object, instead.

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 12, 2021. https://www.bluetooth.com/.

See Also

Objects
`bluetoothLENode`

Functions
`addTrafficSource` | `pushChannelData` | `runNode` | `statistics` | `updateChannelList`

configureBIG

Configure BIG parameters at Bluetooth LE isochronous broadcaster and receiver

Syntax

```
cfgBluetoothLEBIG = configureBIG(cfgBluetoothLEBIG,isoBroadcasterNode,
receiverNode)
```

Description

`cfgBluetoothLEBIG = configureBIG(cfgBluetoothLEBIG,isoBroadcasterNode, receiverNode)` configures the broadcast isochronous group (BIG) parameters at the Bluetooth low energy (LE) isochronous broadcaster node `isoBroadcasterNode` and receiver node `receiverNode` by using the Bluetooth LE BIG configuration object `cfgBluetoothLEBIG`.

Examples

Create and Configure Bluetooth LE Node with BIG Configuration Properties

Create a Bluetooth LE node, specifying the role as `"isochronous-broadcaster"`.

```
isoBroadcasterNode = bluetoothLENode("isochronous-broadcaster");
```

Create a Bluetooth LE node, specifying the role as `"synchronized-receiver"`.

```
receiverNode = bluetoothLENode("synchronized-receiver");
```

Create a default BIG configuration object.

```
bigConfig = bluetoothLEBIGConfig;
```

Specify the number of BISes in the BIG, the number of subevents in each BIS event in the BIG, and the BIS arrangement.

```
bigConfig.NumBIS = 2;
bigConfig.NumSubevents = 2;
bigConfig.BISArrangement = "interleaved";
```

Specify the number of payloads associated with a BIS event.

```
bigConfig.BurstNumber = 2;
```

Specify the time interval between successive BIS subevents and the isochronous event interval.

```
bigConfig.SubInterval = 0.006; % In seconds
bigConfig.ISOInterval = 0.015  % In seconds
```

```
bigConfig =
  bluetoothLEBIGConfig with properties:

        SeedAccessAddress: "78E52493"
```

```
                PHYMode: "LE1M"
                 NumBIS: 2
            ISOInterval: 0.0150
             BISSpacing: 0.0022
            SubInterval: 0.0060
                 MaxPDU: 251
            BurstNumber: 2
    PretransmissionOffset: 0
          RepetitionCount: 1
             NumSubevents: 2
           BISArrangement: "interleaved"
                BIGOffset: 0
        ReceiveBISNumbers: 1
             UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ... ]
            InstantOffset: 6
    BaseCRCInitialization: "1234"
```

Assign the BIG configuration to the Bluetooth LE nodes.

```
configureBIG(bigConfig,isoBroadcasterNode,receiverNode);
```

Input Arguments

`cfgBluetoothLEBIG` — Bluetooth LE BIG configuration object
`bluetoothLEBIGConfig` object

Bluetooth LE BIG configuration object, specified as a `bluetoothLEBIGConfig` object.

`isoBroadcasterNode` — Bluetooth LE node object for the isochronous broadcaster
`bluetoothLENode` object

Bluetooth LE node object for the isochronous broadcaster, specified as a `bluetoothLENode` object with the Role property set to `"isochronous-broadcaster"`.

`receiverNode` — Bluetooth LE node object for the synchronized receiver
`bluetoothLENode` object

Bluetooth LE node object for the synchronized receiver, specified as a `bluetoothLENode` object with the Role property set to `"synchronized-receiver"`.

Output Arguments

`cfgBluetoothLEBIG` — Bluetooth LE BIG configuration object
`bluetoothLEBIGConfig` object

Bluetooth LE BIG configuration object, returned as a `bluetoothLEBIGConfig` object.

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 12, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Objects

`bluetoothLENode` | `bluetoothLEConnectionConfig`

configureConnection

Configure connection between Bluetooth BR Central and Peripheral nodes

Syntax

```
connection = configureConnection(cfgBluetoothConnection,centralNode,
peripheralNode)
```

Description

`connection = configureConnection(cfgBluetoothConnection,centralNode, peripheralNode)` configures the connection between the Bluetooth basic rate (BR) Central node, `centralNode`, and the Peripheral node, `peripheralNode`, by using the Bluetooth BR connection configuration object `cfgBluetoothConnection`.

Examples

Create, Configure, and Simulate Bluetooth BR Nodes

Initialize the wireless network simulator.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and the other with the `"peripheral"` role. Specify the position of the Peripheral node, in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between the Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure the connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode)
```

```
connection =
  bluetoothConnectionConfig with properties:

    CentralToPeripheralACLPacketType: "DH1"
    PeripheralToCentralACLPacketType: "DH1"
                        SCOPacketType: "None"
                  HoppingSequenceType: "Connection adaptive"
                         UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 ... ]
                         PollInterval: 40
                        InstantOffset: 240
                     TransmitterPower: 20
                   SupervisionTimeout: 32000
```

```
   Read-only properties:
                      CentralAddress: "D091BBE70001"
                    PrimaryLTAddress: 1
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
    GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Specify the simulation time, in seconds.

```
simulationTime = 0.3;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve the application, baseband, and physical layer (PHY) statistics corresponding to the Central and Peripheral nodes.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
        Name: "Node1"
          ID: 1
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

```
peripheralStats = statistics(peripheralNode)
```

```
peripheralStats = struct with fields:
        Name: "Node2"
          ID: 2
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

Input Arguments

`connection` — Bluetooth BR connection configuration object

`bluetoothConnectionConfig` object

Bluetooth BR connection configuration object, specified as a `bluetoothConnectionConfig` object.

`centralNode` — Bluetooth BR node object for Central node
`bluetoothNode` object

Bluetooth BR node object for the Central node, specified as a `bluetoothNode` object with the Role property set to `"central"`.

`peripheralNode` — Bluetooth BR node object for Peripheral node
`bluetoothNode` object

Bluetooth BR node object for the Peripheral node, specified as a `bluetoothNode` object with the Role property set to `"peripheral"`.

Output Arguments

`cfgBluetoothConnection` — Bluetooth BR connection configuration object
`bluetoothConnectionConfig` object

Bluetooth BR connection configuration object, returned as a `bluetoothConnectionConfig` object.

Version History

Introduced in R2022b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed May 22, 2022. https://www.bluetooth.com/.

[2] Bluetooth Core Specifications Working Group. "Bluetooth Core Specification" v5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothConnectionConfig` | `bluetoothNode`

configureConnection

Configure LL connection between Bluetooth LE Central and Peripheral nodes

Syntax

```
cfgBluetoothLEConnection = configureConnection(cfgBluetoothLEConnection,
centralNode,peripheralNode)
```

Description

`cfgBluetoothLEConnection = configureConnection(cfgBluetoothLEConnection, centralNode,peripheralNode)` configures the link layer (LL) connection between the Bluetooth low energy (LE) Central node `centralNode` and Peripheral node `peripheralNode` by using the Bluetooth LE LL connection configuration object `cfgBluetoothLEConnection`.

Examples

Create and Configure LL Connection Between Bluetooth LE Nodes

Create a Bluetooth LE node, specifying the role as `"central"`.

```
centralNode = bluetoothLENode("central");
```

Create two Bluetooth LE nodes, specifying the role as `"peripheral"`.

```
peripheralNode1 = bluetoothLENode("peripheral");
peripheralNode2 = bluetoothLENode("peripheral");
```

Create a default Bluetooth LE configuration object to share connection between the Central and Peripheral nodes.

```
connectionConfig = bluetoothLEConnectionConfig
```

```
connectionConfig =
  bluetoothLEConnectionConfig with properties:

    ConnectionInterval: 0.0200
         AccessAddress: "5DA44270"
          UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
             Algorithm: 1
          HopIncrement: 5
     CRCInitialization: "012345"
    SupervisionTimeout: 1
               PHYMode: "LE1M"
         InstantOffset: 6
      ConnectionOffset: 0
          ActivePeriod: 0.0200
```

Specify the connection interval, connection offset, and active period for the LL connection.

```
connectionConfig.ConnectionInterval = 0.04; % In seconds
connectionConfig.ConnectionOffset = 0;      % In seconds
connectionConfig.ActivePeriod = 0.02;       % In seconds
```

Specify the unique connection address for the LL connection.

```
connectionConfig.AccessAddress = "5DA44271"
```

```
connectionConfig =
  bluetoothLEConnectionConfig with properties:

    ConnectionInterval: 0.0400
         AccessAddress: "5DA44271"
          UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
             Algorithm: 1
          HopIncrement: 5
     CRCInitialization: "012345"
    SupervisionTimeout: 1
               PHYMode: "LE1M"
         InstantOffset: 6
      ConnectionOffset: 0
          ActivePeriod: 0.0200
```

Configure the LL connection between the Central node and Peripheral node 1.

```
configureConnection(connectionConfig,centralNode,peripheralNode1);
```

Update the LL connection configuration object to share connection parameters between the Central node and Peripheral node 2.

```
connectionConfig.ConnectionOffset = 0.02;   % In seconds
connectionConfig.ActivePeriod = 0.02;       % In seconds
connectionConfig.AccessAddress = "5DA44272"
```

```
connectionConfig =
  bluetoothLEConnectionConfig with properties:

    ConnectionInterval: 0.0400
         AccessAddress: "5DA44272"
          UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
             Algorithm: 1
          HopIncrement: 5
     CRCInitialization: "012345"
    SupervisionTimeout: 1
               PHYMode: "LE1M"
         InstantOffset: 6
      ConnectionOffset: 0.0200
          ActivePeriod: 0.0200
```

Configure the LL connection between the Central node and Peripheral node 2.

```
configureConnection(connectionConfig,centralNode,peripheralNode2);
```

Input Arguments

`cfgBluetoothLEConnection` — Bluetooth LE connection configuration object
`bluetoothLEConnectionConfig` object

Bluetooth LE connection configuration object, specified as a `bluetoothLEConnectionConfig` object.

`centralNode` — Bluetooth LE node object for the Central node
`bluetoothLENode` object

Bluetooth LE node object for the Central node, specified as a `bluetoothLENode` object with the Role property set to `"central"`.

`peripheralNode` — Bluetooth LE node object for the Peripheral node
`bluetoothLENode` object

Bluetooth LE node object for the Peripheral node, specified as a `bluetoothLENode` object with the Role property set to `"peripheral"`.

Output Arguments

`cfgBluetoothLEConnection` — Bluetooth LE connection configuration object
`bluetoothLEConnectionConfig` object

Bluetooth LE connection configuration object, returned as a `bluetoothLEConnectionConfig` object.

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 12, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothLENode` | `bluetoothLEBIGConfig` | `bluetoothMeshProfileConfig` | `bluetoothMeshFriendshipConfig`

configureFriendship

Configure friendship between Friend node and LPN

Syntax

```
cfgBluetoothMeshFriendship = configureFriendship(cfgBluetoothMeshFriendship,
friendNode,lowPowerNode)
```

Description

`cfgBluetoothMeshFriendship = configureFriendship(cfgBluetoothMeshFriendship, friendNode,lowPowerNode)` establishes friendship between a Friend node, `friendNode`, and a low power node (LPN), `lowPowerNode`, by configuring the friendship parameters using the Bluetooth mesh friendship configuration object, `cfgBluetoothMeshFriendship`. At the Friend node, friendship with multiple LPNs is not supported.

Examples

Create and Configure Bluetooth Mesh Node with Friendship Properties

Create a Bluetooth mesh profile configuration object, specifying the element address and enabling the Friend feature of the Friend node.

```
meshFriendshipCfg = bluetoothMeshProfileConfig(ElementAddress="000A",Friend=true)
```

```
meshFriendshipCfg =
  bluetoothMeshProfileConfig with properties:

             ElementAddress: "000A"
                      Relay: 0
                     Friend: 1
                   LowPower: 0
       NetworkTransmissions: 1
    NetworkTransmitInterval: 0.0100
                        TTL: 127
```

Create a Bluetooth mesh profile configuration object for an LPN, specifying the element address and enabling the low power feature of the LPN.

```
meshLPNCfg = bluetoothMeshProfileConfig(ElementAddress="000F",LowPower=true)
```

```
meshLPNCfg =
  bluetoothMeshProfileConfig with properties:

             ElementAddress: "000F"
                      Relay: 0
                     Friend: 0
                   LowPower: 1
       NetworkTransmissions: 1
    NetworkTransmitInterval: 0.0100
```

```
                    TTL: 127
```

Create a Bluetooth mesh node with the friend feature enabled. Specify the role as `"broadcaster-observer"` and assign the mesh profile configuration.

```
meshFriendNode = bluetoothLENode("broadcaster-observer",MeshConfig=meshFriendshipCfg);
```

Create a Bluetooth mesh node with the low power feature enabled. Specify the role as `"broadcaster-observer"` and assign the mesh profile configuration.

```
meshLPN = bluetoothLENode("broadcaster-observer",MeshConfig=meshLPNCfg);
```

Create a default Bluetooth mesh friendship configuration object. Specify the poll timeout, receive window supported by the Friend node, and receive delay requested by the LPN.

```
friendshipConfig = bluetoothMeshFriendshipConfig;
friendshipConfig.PollTimeout = 3;                       % In seconds
friendshipConfig.ReceiveWindow = 0.180;                 % In seconds
friendshipConfig.ReceiveDelay = 0.05                    % In seconds
```

```
friendshipConfig =
  bluetoothMeshFriendshipConfig with properties:

     ReceiveDelay: 0.0500
    ReceiveWindow: 0.1800
      PollTimeout: 3
```

Configure friendship between the Friend node and LPN.

```
configureFriendship(friendshipConfig,meshFriendNode,meshLPN);
```

Input Arguments

`cfgBluetoothMeshFriendship` — Bluetooth mesh friendship configuration object

`bluetoothMeshFriendshipConfig` object

Bluetooth mesh friendship configuration object, specified as a `bluetoothMeshFriendshipConfig` object.

`friendNode` — Bluetooth LE node object for broadcaster observer

`bluetoothLENode` object

Bluetooth LE node object for the broadcaster observer, specified as a `bluetoothLENode` object with the Role property set to `"broadcaster-observer"`. You must also set the Friend property of the `bluetoothMeshProfileConfig` object to `true` and the MeshConfig property of the `bluetoothLENode` object to `bluetoothMeshProfileConfig` object.

`lowPowerNode` — Bluetooth LE node object for broadcaster observer

`bluetoothLENode` object

Bluetooth LE node object for the broadcaster observer, specified as a `bluetoothLENode` object with the Role property set to `"broadcaster-observer"`. You must also set the LowPower property of the `bluetoothMeshProfileConfig` object to `true` and the MeshConfig property of the `bluetoothLENode` object to `bluetoothMeshProfileConfig` object.

Output Arguments

`cfgBluetoothMeshFriendship` — Bluetooth mesh friendship configuration object
`bluetoothMeshFriendshipConfig` object

Bluetooth mesh friendship configuration object, returned as a `bluetoothMeshFriendshipConfig` object.

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 12, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[3] Bluetooth Special Interest Group (SIG). "Bluetooth Mesh Profile". v1.0.1. https://www.bluetooth.com/.

See Also

Objects
`bluetoothLENode` | `bluetoothLEConnectionConfig` | `bluetoothMeshProfileConfig` | `bluetoothMeshFriendshipConfig`

generate

Generate next On-Off application traffic packet

Syntax

```
[dt,packetSize] = generate(cfgOnOff)
[dt,packetSize] = generate(cfgOnOff,elapsedTime)
[___,packet] = generate(___)
```

Description

`[dt,packetSize] = generate(cfgOnOff)` generates the next On-Off application traffic packet based on the specified configuration object, `cfgOnOff`. The object function returns the time remaining to generate the next packet, `dt`, and the size of the current packet, `packetSize`.

`[dt,packetSize] = generate(cfgOnOff,elapsedTime)` specifies the time elapsed, `elapsedTime`, since the previous call of this object function.

`[___,packet] = generate(___)` returns an On-Off application traffic packet. Specify an argument combination from any of the previous syntaxes.

Examples

Generate On-Off Application Traffic Pattern and Data Packet

Create an On-Off application traffic pattern object to generate an On-Off data packet.

```
cfgOnOff = networkTrafficOnOff('GeneratePacket',true);
```

Generate an On-Off application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgOnOff);
```

Input Arguments

`cfgOnOff` — Configuration object to generate On-Off application traffic pattern

`networkTrafficOnOff` object

Configuration object to generate an On-Off application traffic pattern, specified as a `networkTrafficOnOff` object.

`elapsedTime` — Time elapsed since previous call of this object function

nonnegative scalar

Time elapsed since the previous call of this object function, specified as a nonnegative scalar. Units are in milliseconds.

Data Types: `double`

Output Arguments

`dt` — Time remaining to generate next packet

nonnegative scalar

Time remaining to generate the next packet, returned as a nonnegative scalar. Units are in milliseconds.

Data Types: `double`

`packetSize` — Size of current packet

positive scalar

Size of the current packet, returned as a positive scalar. Units are in bytes.

Data Types: `double`

`packet` — Application data packet

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

Application data packet, returned as a column vector of integers in the range [0, 255]. This output contains application data that the `ApplicationData` property of the input `cfgOnOff` specifies. If you do not specify `ApplicationData`, this output is a column vector of 1s.

Dependencies

To enable this output argument, set the `GeneratePacket` property of the `cfg` input to `1` (`true`).

Data Types: `double`

Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`networkTrafficOnOff`

getElementPosition

Return positions of array elements

Syntax

```
positions = getElementPosition(cfg)
```

Description

`positions = getElementPosition(cfg)` returns the positions of the array elements for the input Bluetooth low energy (LE) angle estimation configuration object. The object function uses the origin of the local coordinate system as the first position of the first antenna of the antenna array.

Examples

Return Positions of Antenna Array Elements

Create a default Bluetooth LE angle estimation configuration object.

```
cfgAngle = bleAngleEstimateConfig
```

```
cfgAngle =
  bleAngleEstimateConfig with properties:

            ArraySize: 4
       ElementSpacing: 0.5000
    EnableCustomArray: 0
         SlotDuration: 2
     SwitchingPattern: [1 2 3 4]
```

Return the positions of antenna array elements.

```
positions = getElementPosition(cfgAngle)
```

```
positions = 3×4

         0         0         0         0
         0    0.5000    1.0000    1.5000
         0         0         0         0
```

Input Arguments

`cfg` — Bluetooth LE angle estimation configuration object

`bleAngleEstimateConfig` object

Bluetooth LE angle estimation configuration object, specified as a `bleAngleEstimateConfig` object.

Output Arguments

`positions` — Positions of array elements
3-by-*N* matrix

Positions of array elements, returned as a 3-by-*N* matrix, where *N* is the number of array elements specified by the `cfg` input. Each column in `pos` defines the position of the array element in the form [*x*; *y*; *z*] in the local coordinate system.

Data Types: `double`

Version History

Introduced in R2020b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[3] Wooley, Martin. *Bluetooth Direction Finding: A Technical Overview*. Bluetooth Special Interest Group (SIG), Accessed December 6, 2021, https://www.bluetooth.com/.

Extended Capabilities

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

See Also

Functions
`bleAngleEstimate` | `getNumElements` | `bleWaveformGenerator` | `bleIdealReceiver`

Objects
`bleAngleEstimateConfig`

getNumElements

Return number of elements in array

Syntax

```
numElements = getNumElements(cfgAngle)
```

Description

`numElements = getNumElements(cfgAngle)` returns the number of elements in the array for the input Bluetooth low energy (LE) angle estimation configuration object.

Examples

Return Number of Elements in Antenna Array

Create a Bluetooth LE angle estimation configuration object, specifying a 2-by-3 URA.

```
cfgAngle = bleAngleEstimateConfig('ArraySize',[2 3])
```

```
cfgAngle =
  bleAngleEstimateConfig with properties:

            ArraySize: [2 3]
       ElementSpacing: 0.5000
    EnableCustomArray: 0
         SlotDuration: 2
     SwitchingPattern: [1 2 3 4]
```

Return the number of elements in the antenna array.

```
numElements = getNumElements(cfgAngle)
```

```
numElements = 6
```

Input Arguments

`cfgAngle` — Bluetooth LE angle estimation configuration object

`bleAngleEstimateConfig` object

Bluetooth LE angle estimation configuration object, specified as a `bleAngleEstimateConfig` object.

Output Arguments

`numElements` — Number of elements in array

positive integer

Number of elements in the array, returned as a positive integer. If the SlotDuration property of the `cfg` input is 2 µs, the value of this output is in the range [2, 38]. If the `SlotDuration` property is 1 µs, the value of this output is in the range [2, 75].

Data Types: `double`

Version History

Introduced in R2020b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

[3] Wooley, Martin. *Bluetooth Direction Finding: A Technical Overview*. Bluetooth Special Interest Group (SIG), Accessed April 6, 2020, https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bleAngleEstimate` | `getElementPosition` | `bleWaveformGenerator` | `bleIdealReceiver`

Objects

`bleAngleEstimateConfig`

getPayloadLength

Return payload length for Bluetooth BR/EDR format configuration

Syntax

```
payloadLength = getPayloadLength(cfg)
```

Description

`payloadLength = getPayloadLength(cfg)` returns the payload length, in bytes, for the Bluetooth BR/EDR format configuration, `cfg`.

Examples

Get Payload Length for Bluetooth BR/EDR Format Configuration

Create a default Bluetooth BR/EDR waveform configuration object.

```
cfgPayload = bluetoothWaveformConfig
```

```
cfgPayload =
  bluetoothWaveformConfig with properties:

                      Mode: 'BR'
                PacketType: 'FHS'
             DeviceAddress: '0123456789AB'
    LogicalTransportAddress: [3x1 double]
          HeaderControlBits: [3x1 double]
            ModulationIndex: 0.3200
           SamplesPerSymbol: 8
               WhitenStatus: 'On'
       WhitenInitialization: [7x1 double]
```

Get the payload length of the default `'FHS'` packet.

```
payloadLength = getPayloadLength(cfgPayload)
```

```
payloadLength = 18
```

Create another Bluetooth BR/EDR waveform configuration object, specifying the packet type as `'HV1'`.

```
cfgPayload1 = bluetoothWaveformConfig('PacketType','HV1');
```

Get the payload length of the specified `'HV1'` packet.

```
payloadLength1 = getPayloadLength(cfgPayload1)
```

```
payloadLength1 = 10
```

Input Arguments

`cfg` — Bluetooth BR/EDR format configuration

`bluetoothWaveformConfig` object

Bluetooth BR/EDR format configuration, specified as `bluetoothWaveformConfig` object.

Output Arguments

`payloadLength` — Payload length of packet

18 (default) | nonnegative integer

Payload length of packet, returned as a nonnegative integer. This value indicates the number of bytes to be processed in a packet.

Data Types: `double`

Version History

Introduced in R2020a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bluetoothWaveformGenerator` | `bluetoothIdealReceiver`

Objects

`bluetoothPhyConfig` | `bluetoothWaveformConfig`

getPhyConfigProperties

Return updated properties of Bluetooth BR/EDR PHY configuration object

Syntax

```
cfgPHY = getPhyConfigProperties(cfg)
```

Description

`cfgPHY = getPhyConfigProperties(cfg)` returns updated Bluetooth BR/EDR PHY configuration properties, `cfgPHY`, for the Bluetooth BR/EDR waveform format configuration object, `cfg`.

Examples

Get PHY Configuration Properties of Bluetooth BR/EDR PHY Configuration Object

Create a default Bluetooth BR/EDR waveform configuration object.

```
cfgWaveform = bluetoothWaveformConfig
```

```
cfgWaveform =
  bluetoothWaveformConfig with properties:

                      Mode: 'BR'
                PacketType: 'FHS'
             DeviceAddress: '0123456789AB'
    LogicalTransportAddress: [3x1 double]
          HeaderControlBits: [3x1 double]
            ModulationIndex: 0.3200
           SamplesPerSymbol: 8
               WhitenStatus: 'On'
       WhitenInitialization: [7x1 double]
```

Get the PHY configuration properties for the created Bluetooth BR/EDR waveform configuration object.

```
cfgPHY = getPhyConfigProperties (cfgWaveform)
```

```
cfgPHY =
  bluetoothPhyConfig with properties:

                    Mode: 'BR'
           DeviceAddress: '0123456789AB'
         ModulationIndex: 0.3200
        SamplesPerSymbol: 8
            WhitenStatus: 'On'
    WhitenInitialization: [7x1 double]
```

Create another Bluetooth BR/EDR waveform configuration object, specifying the PHY transmission mode as `'EDR2M'`.

```
cfgWaveform2 = bluetoothWaveformConfig('Mode','EDR2M');
```

Get the PHY configuration properties for this Bluetooth BR/EDR waveform configuration object.

```
cfgPHY2 = getPhyConfigProperties (cfgWaveform2)
```

```
cfgPHY2 =
  bluetoothPhyConfig with properties:

                    Mode: 'EDR2M'
           DeviceAddress: '0123456789AB'
         ModulationIndex: 0.3200
        SamplesPerSymbol: 8
            WhitenStatus: 'On'
    WhitenInitialization: [7x1 double]
```

Input Arguments

`cfg` — Bluetooth BR/EDR format configuration

`bluetoothWaveformConfig` object

Bluetooth BR/EDR Format configuration, specified as `bluetoothWaveformConfig` object.

Output Arguments

`cfgPHY` — Configuration object for Bluetooth BR/EDR PHY

`bluetoothPHYConfig` object

Configuration object for Bluetooth BR/EDR PHY, returned as a `bluetoothPhyConfig` object.

Version History

Introduced in R2020a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`bluetoothWaveformGenerator` | `bluetoothIdealReceiver`

Objects
`bluetoothPhyConfig` | `bluetoothWaveformConfig`

nextHop

Select Bluetooth BR/EDR channel index to hop for next frequency

Syntax

```
[channelIndex,X] = nextHop(cfgFH,Clock)
```

Description

`[channelIndex,X] = nextHop(cfgFH,Clock)` selects a Bluetooth basic rate/enhanced data rate (BR/EDR) channel index, `channelIndex` to hop for next frequency. This selection is based on the Bluetooth BR/EDR frequency hopping object, `FH`, the clock, `Clock`, and the “SequenceType” on page 2-0 property of the “cfgFH” on page 3-0 . The object function also returns `X`, which is required for implementing whitening process in the physical layer (PHY).

Examples

Select Bluetooth BR/EDR Channel Index Using Default Values

Create a default Bluetooth BR/EDR channel index object for frequency hopping.

```
cfgFH = bluetoothFrequencyHop
```

```
cfgFH =
  bluetoothFrequencyHop with properties:

    DeviceAddress: '9E8B33'
     SequenceType: 'Inquiry'
           KNudge: 0
          KOffset: 24
```

Specify a clock value.

```
inputClock = '12C';        % 28-bit
```

Select a Bluetooth BR/EDR channel index to hop for the next frequency.

```
[channelIndex,X] = nextHop(cfgFH,inputClock)
```

```
channelIndex = 41
```

```
X = 30
```

Input Arguments

`cfgFH` — Bluetooth BR/EDR channel index for frequency hopping

`bluetoothFrequencyHop` object

Bluetooth BR/EDR channel index for frequency hopping, specified as a `bluetoothFrequencyHop` object.

`Clock` — Clock
character vector in hexadecimal format | string scalar in hexadecimal format | numeric scalar in the range [0, 2^{28}–1]

Clock, specified as one of these values:

- Character vector — This vector represents the `Clock` in hexadecimal format
- String scalar — This scalar represents the `Clock` in hexadecimal format
- Numeric scalar — This scalar represent the `Clock` in the range [0, 2^{28}–1]

This argument is a 28-bit value that computes the inputs to the hop selection kernel. This table shows the dependency of this argument on the value of the SequenceType property of the `FH` input.

Value of SequenceType Property	`Clock` Input
`'Connection basic'` or `'Connection adaptive'`	Indicates native clock of the Central
`'Page'` or `'Inquiry'`	Indicates native clock of the Peripheral
`'Page scan'` or `'Inquiry scan'`	Indicates the estimated value of the clock for the Peripheral
`'Peripheral page response'`	Indicates the value when the access code of the recipient is detected
`'Central page response'`	Indicates the value that triggered a response from the paged device

Data Types: `char` | `string` | `double`

Note From R2022a, this object function uses 'Central' and 'Peripheral' terminologies to represent 'Master' and 'Slave' nodes, respectively.

Output Arguments

`channelIndex` — Channel index
integer in the range [0, 78]

Channel index, returned as an integer in the range [0, 78].

Data Types: `double`

`X` — Control signal to be used in whitening process
nonnegative integer

Control signal to be used in whitening process, returned as a nonnegative integer.

Data Types: `double`

Version History

Introduced in R2020b

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`bluetoothFrequencyHop` | `bleChannelSelection`

pathLoss

Compute path loss and received signal power

Syntax

```
[pl,rxPower] = pathLoss(cfgRange)
```

Description

`[pl,rxPower] = pathLoss(cfgRange)` returns the path loss, `pl`, in dB and received power, `rxPower`, in dBm for the specified Bluetooth basic rate/enhanced data rate (BR/EDR) or low energy (LE) range estimation configuration object, `cfgRange`.

Examples

Compute Path Loss and Received Power Between Two Bluetooth BR Devices

Create a Bluetooth BR/EDR or LE range estimation configuration, specifying the physical layer (PHY) transmission mode as "BR", transmitter output power as 10, and receiver sensitivity as -72.

```
cfgRange = bluetoothRangeConfig(Mode="BR",TransmitterPower=10,ReceiverSensitivity=-72)
```

```
cfgRange =
  bluetoothRangeConfig with properties:

                Environment: 'Outdoor'
             SignalPowerType: 'ReceiverSensitivity'
                        Mode: 'BR'
          ReceiverSensitivity: -72
                    LinkMargin: 15
             TransmitterPower: 10
       TransmitterAntennaGain: 0
          ReceiverAntennaGain: 0
         TransmitterCableLoss: 1.2500
            ReceiverCableLoss: 1.2500
     TransmitterAntennaHeight: 1
        ReceiverAntennaHeight: 1

   Read-only properties:
                FSPLDistance: 16.4188
               PathLossModel: 'TwoRayGroundReflection'
```

Compute path loss and received power between two Bluetooth BR devices.

```
[pl,rxPower] = pathLoss(cfgRange)
```

```
pl = 64.5000
```

```
rxPower = -57
```

Input Arguments

`cfgRange` — Bluetooth BR/EDR or LE range estimation configuration parameters

`bluetoothRangeConfig` object

Bluetooth BR/EDR or LE range estimation configuration parameters, specified as a `bluetoothRangeConfig` object.

Output Arguments

`pl` — Path loss between two Bluetooth BR/EDR or LE devices

scalar

Path loss between two Bluetooth BR/EDR or LE devices, returned as a scalar. Units are in dB.

Data Types: `double`

`rxPower` — Received signal power between two Bluetooth BR/EDR or LE devices

scalar

Received signal power between two Bluetooth BR/EDR or LE devices, returned as a scalar. Units are in dBm.

Data Types: `double`

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 12, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification." Version 5.3. https://www.bluetooth.com/.

Extended Capabilities

C/C++ Code Generation

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

See Also

Objects

`bluetoothRangeConfig`

write

Write protocol packet data to PCAP or PCAPNG file

Syntax

```
write(pcapObj,packet,timestamp)
write(pcapngObj,packet,timestamp,interfaceID)
write( ___ ,Name,Value)
```

Description

`write(pcapObj,packet,timestamp)` writes the protocol packet data to the PCAP file specified in the PCAP file writer object, `pcapObj`. Input `packet` specifies the protocol packet and input `timestamp` specifies the packet arrival time.

`write(pcapngObj,packet,timestamp,interfaceID)` writes protocol packet data to a PCAPNG file specified in the PCAPNG file writer object, `pcapngObj`. Input `packet`, `timestamp`, and `interfaceID` specifies the protocol packet, packet arrival time, and interface identifier, respectively.

`write( ___ ,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input argument combinations from any of the previous syntaxes. For example, `'PacketFormat','bits'` sets the format of the protocol packets to bits.

Examples

Write Bluetooth LE Packet Data to PCAP File

Create a PCAP file writer object, specifying the name of the PCAP file. Specify the Bluetooth low energy (LE) link type.

```
pcapObj = pcapWriter('FileName','writeBluetoothLEpacket');
bleLinkType = 251;
```

Write a global header to the PCAP file.

```
writeGlobalHeader(pcapObj,bleLinkType);
```

Specify the Bluetooth LE link layer (LL) packet.

```
llpacket = '42BC13E206120E00050014010A001F00400017001700000007D47C0';
```

Write Bluetooth LE LL packet to the PCAP file.

```
timestamp = 129100;                 % Packet arrival time in POSIX® microseconds elapsed since 1/
write(pcapObj,llpacket,timestamp);
```

Write Bluetooth LE Packet Data to PCAPNG File

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapngObj = pcapngWriter('FileName','writeBluetoothLEpacket');
```

Write an interface description block for Bluetooth LE.

```
interfaceName = 'Bluetooth LE interface';
bleLinkType = 251;
interfaceId = writeInterfaceDescriptionBlock(pcapngObj,bleLinkType, ...
    interfaceName);
```

Specify the Bluetooth LE LL packet.

```
llpacket = '42BC13E206120E00050014010A001F00400017001700000007D47C0';
```

Write Bluetooth LE LL packet to the PCAPNG format file.

```
timestamp = 0;                                  % Packet arrival time in POSIX® microseconds elap
packetComment = 'This is Bluetooth LE packet';
write(pcapngObj, llpacket,timestamp,interfaceId,'PacketComment', ...
    packetComment);
```

Input Arguments

Note The `pcapWriter` and `pcapngWriter` objects do not overwrite the existing PCAP or PCAPNG files, respectively. Each time when you create these objects, specify a unique PCAP or PCAPNG file name.

`pcapObj` — PCAP file writer object

`pcapWriter` object

PCAP file writer object, specified as a `pcapWriter` object.

`packet` — Protocol packet

binary-valued vector | character vector | string scalar | numeric vector | *n*-by-2 character array

Protocol packet, specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format .

Data Types: `char` | `string` | `double`

`timestamp` — Packet arrival time

nonnegative integer

Packet arrival time in POSIX microseconds elapsed since 1/1/1970, specified as a nonnegative integer.

Data Types: `double`

`pcapngObj` — PCAPNG file writer object
`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`interfaceID` — Unique identifier for an interface
nonnegative scalar

Unique identifier for an interface, specified as a nonnegative scalar.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'PacketFormat','bits'` sets the format of the protocol packets to bits.

`PacketFormat` — Format of the protocol packet
`'octets'` (default) | `'bits'`

Format of the protocol packet, specified as the comma-separated pair consisting of `PacketFormat` and `'octets'` or `'bits'`. If this value is specified as `'octets'`, `packet` is specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format .

Data Types: `char` | `string` | `double`

`PacketComment` — Comment for protocol packet
`''` (default) | character vector | string scalar

Comment for the protocol packet, specified as the comma-separated pair consisting of `PacketComment` and a character vector or a string scalar.

Dependencies

To enable this name-value pair argument, specify the `pcapngObj` input argument.

Data Types: `char` | `string`

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`writeGlobalHeader` | `writeCustomBlock` | `writeInterfaceDescriptionBlock`

Objects

`pcapWriter` | `pcapngWriter`

pushChannelData

(To be removed) Push data from channel to reception buffer

Note will be removed in a future release. To push data from the channel to the reception buffer of the Bluetooth LE node, use the `wirelessNetworkSimulator` object, instead.

Syntax

```
pushChannelData(bluetoothLENodeObj,packet)
```

Description

`pushChannelData(bluetoothLENodeObj,packet)` pushes the packet `packet` from the channel to the reception buffer of the Bluetooth low energy (LE) node specified by the `bluetoothLENodeObj` object.

Examples

Run Bluetooth LE Node

Create a Bluetooth LE node, specifying the role as `"central"`.

```
centralNode = bluetoothLENode("central");
```

Create a Bluetooth LE node, specifying the role as `"peripheral"`.

```
peripheralNode = bluetoothLENode("peripheral");
```

Create a default Bluetooth LE configuration object to establish a connection between the Central and Peripheral nodes.

```
cfgConnection = bluetoothLEConnectionConfig;
```

Configure the connection between the Central and Peripheral nodes.

```
configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Specify the data rate in kb/s and the packet size in bytes. Generate an application packet with a payload by enabling packet generation. Add application traffic from the Central to the Peripheral node by using the `addTrafficSource` object function.

```
traffic = networkTrafficOnOff(DataRate=50, ...
    PacketSize=20, ...
    GeneratePacket=true);
addTrafficSource(centralNode,traffic,"DestinationNode",peripheralNode.Name);
```

Create a Bluetooth LE network consisting of a Central and a Peripheral node.

```
nodes = {centralNode peripheralNode};
numNodes = length(nodes);
```

Set the current simulation time in seconds.

```
currentTime = 0;
```

Run the Bluetooth LE nodes at the current simulation time. The `runNode` object function returns the time instant at which the node runs again.

```
for index = 1:numNodes
    nextInvokeTime = runNode(nodes{index},currentTime);
end
```

If the transmit buffer has packet to be transmitted, the `channelInvokeDecision` object function performs these tasks.

- Determines whether the receiving node wants to receive the packet.
- Retrieves the receiver information required to apply the channel to the transmitted packet.

If the node wants to receive the packet, apply the channel to the transmitted packet and push the data from the channel to the reception buffer by using the `pushChannelData` object function.

```
for txIndex = 1:numNodes
    txNode = nodes{txIndex};
    if (txNode.TransmitBuffer.Type ~= 0)
        txPacket = txNode.TransmitBuffer;
        for rxIndex = 1:numNodes
            rxNode = nodes{rxIndex};
            [flag,rxInfo] = channelInvokeDecision(rxNode,txPacket);
            if flag
                % Apply a path loss to the transmitted data in this comment
                % line before pushing the data to the reception buffer
                pushChannelData(rxNode,txPacket);
            end
        end
    end
end
```

Input Arguments

`bluetoothLENodeObj` — Bluetooth LE node object

`bluetoothLENode` object

Bluetooth LE node object, specified as a `bluetoothLENode` object.

`packet` — Packet received from channel

structure

Packet received from the channel, specified as a structure with these fields.

Structure Field	Structure Value	Description
`Type`	`0`, `1`, `2`, or `3`	Type of the incoming signal. Each value represents one of these packet types. • `0` — Invalid packet • `1` — WLAN packet • `2` — 5G packet • `3` — Bluetooth LE packet The default value is 0.
`TransmitterID`	nonnegative scalar integer	Transmitter node identifier
`TransmitterPosition`	numeric row vector of length three	Position of the transmitter in Cartesian x-, y-, and z-coordinates. The value of this field is in the form [X Y Z]. Units are in meters.
`TransmitterVelocity`	real-valued row vector of length three	Velocity, *v*, of the transmitter in the x- y-, and z-directions. The value of this field is in the form [*vx vy vz*]. Units are in meters per second.
`StartTime`	positive integer	Packet transmission start time at the transmitter or packet arrival time at the receiver. Units are in seconds.
`Duration`	positive integer	Duration of the packet, in seconds
`Power`	scalar	Transmit power of the packet in dBm
`CenterFrequency`	scalar positive integer	Center frequency of the carrier in Hz
`Bandwidth`	scalar positive integer	Carrier bandwidth in Hz
`Abstraction`	`0` (`false`) or `1` (`true`)	Physical layer (PHY) abstraction type. A `1` (`true`) value represents the abstracted PHY. A `0` (`false`) value specifies the full PHY. The default value is `0` (`false`).
`SampleRate`	scalar positive integer	Sample rate of the packet in samples/second

Structure Field	Structure Value	Description
`Data`	matrix	If you set the `Abstraction` field of the `packet` argument, this field specifies the in-phase and quadrature (IQ) samples of the packet as a *M*-by-*N* matrix, where *M* is the number of IQ samples and *N* is the number of Rx antennas. When the `Abstraction` field of the `packet` argument is set to `1` (`true`), this field has frame information.
`Metadata`	structure	Technology specific or abstraction specific information, specified as a structure. The default value is an empty structure.

Data Types: `struct`

Version History

Introduced in R2022a

`pushChannelData` object function will be removed
Not recommended starting in R2022b

`pushChannelData` will be removed in a future release. You can push data from the channel to the reception buffer of the Bluetooth LE node by using the `wirelessNetworkSimulator` object, instead.

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 22, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification". v5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothLENode`

Functions
`addTrafficSource` | `channelInvokeDecision` | `runNode` | `statistics` | `updateChannelList`

runNode

(To be removed) Run Bluetooth LE node

Note will be removed in a future release. To run the Bluetooth LE node, use the `wirelessNetworkSimulator` object, instead.

Syntax

```
nextInvokeTime = runNode(bluetoothLENodeObj,currentTime)
```

Description

`nextInvokeTime = runNode(bluetoothLENodeObj,currentTime)` runs the Bluetooth LE node specified by the `bluetoothLENodeObj` object. The object function runs all the actions scheduled at the current time and returns the time instant `nextInvokeTime` at which the node runs again.

Examples

Run Bluetooth LE Node

Create a Bluetooth LE node, specifying the role as `"central"`.

```
centralNode = bluetoothLENode("central");
```

Create a Bluetooth LE node, specifying the role as `"peripheral"`.

```
peripheralNode = bluetoothLENode("peripheral");
```

Create a default Bluetooth LE configuration object to establish a connection between the Central and Peripheral nodes.

```
cfgConnection = bluetoothLEConnectionConfig;
```

Configure the connection between the Central and Peripheral nodes.

```
configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Specify the data rate in kb/s and the packet size in bytes. Generate an application packet with a payload by enabling packet generation. Add application traffic from the Central to the Peripheral node by using the `addTrafficSource` object function.

```
traffic = networkTrafficOnOff(DataRate=50, ...
    PacketSize=20, ...
    GeneratePacket=true);
addTrafficSource(centralNode,traffic,"DestinationNode",peripheralNode.Name);
```

Create a Bluetooth LE network consisting of a Central and a Peripheral node.

```
nodes = {centralNode peripheralNode};
numNodes = length(nodes);
```

Set the current simulation time in seconds.

```
currentTime = 0;
```

Run the Bluetooth LE nodes at the current simulation time. The `runNode` object function returns the time instant at which the node runs again.

```
for index = 1:numNodes
    nextInvokeTime = runNode(nodes{index},currentTime);
end
```

If the transmit buffer has packet to be transmitted, the `channelInvokeDecision` object function performs these tasks.

- Determines whether the receiving node wants to receive the packet.
- Retrieves the receiver information required to apply the channel to the transmitted packet.

If the node wants to receive the packet, apply the channel to the transmitted packet and push the data from the channel to the reception buffer by using the `pushChannelData` object function.

```
for txIndex = 1:numNodes
    txNode = nodes{txIndex};
    if (txNode.TransmitBuffer.Type ~= 0)
        txPacket = txNode.TransmitBuffer;
        for rxIndex = 1:numNodes
            rxNode = nodes{rxIndex};
            [flag,rxInfo] = channelInvokeDecision(rxNode,txPacket);
            if flag
                % Apply a path loss to the transmitted data in this comment
                % line before pushing the data to the reception buffer
                pushChannelData(rxNode,txPacket);
            end
        end
    end
end
```

Input Arguments

`bluetoothLENodeObj` — Bluetooth LE node object

`bluetoothLENode` object

Bluetooth LE node object, specified as a `bluetoothLENode` object.

`currentTime` — Current simulation time

positive scalar

Current simulation time, specified as a positive scalar. Specify the units in seconds.

Data Types: `double`

Output Arguments

`nextInvokeTime` — Time instant at which Bluetooth LE node runs again
nonnegative scalar

Time instant at which Bluetooth LE node runs again, returned as a nonnegative scalar. Specify the units in seconds.

Data Types: `double`

Version History

Introduced in R2022a

`runNode` object function will be removed
Not recommended starting in R2022b

`runNode` will be removed in a future release. You can run the Bluetooth LE node by using the `wirelessNetworkSimulator` object, instead.

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 12, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification". v5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothLENode`

Functions
`addTrafficSource` | `channelInvokeDecision` | `pushChannelData` | `statistics` | `updateChannelList`

statistics

Get statistics of Bluetooth LE node

Syntax

```
nodeStatistics = statistics(bluetoothLENodeObj)
```

Description

`nodeStatistics = statistics(bluetoothLENodeObj)` returns the statistics of the Bluetooth low energy (LE) node object `bluetoothLENodeObj`.

Examples

Create, Configure, and Simulate Bluetooth LE Network

This example shows you how to simulate a Bluetooth® low energy (LE) network by using Bluetooth® Toolbox.

Using this example, you can:

1. Create and configure a Bluetooth LE piconet with Central and Peripheral nodes.
2. Create and configure a link layer (LL) connection between Central and Peripheral nodes.
3. Add application traffic from the Central to Peripheral nodes.
4. Simulate Bluetooth LE network and retrieve the statistics of the Central and Peripheral nodes.

Create a wireless network simulator.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create a Bluetooth LE node, specifying the role as `"central"`. Specify the name and position of the node.

```
centralNode = bluetoothLENode("central");
centralNode.Name = "CentralNode";
centralNode.Position = [0 0 0];                 % In x-, y-, and z-coordinates in meters
```

Create a Bluetooth LE node, specifying the role as `"peripheral"`. Specify the name and position of the node.

```
peripheralNode = bluetoothLENode("peripheral");
peripheralNode.Name = "PeripheralNode";
peripheralNode.Position = [10 0 0]              % In x-, y-, and z-coordinates in meters
```

```
peripheralNode =
  bluetoothLENode with properties:

        TransmitterPower: 20
         TransmitterGain: 0
           ReceiverRange: 100
```

```
            ReceiverGain: 0
     ReceiverSensitivity: -100
             NoiseFigure: 0
    InterferenceFidelity: 0
                    Name: "PeripheralNode"
                Position: [10 0 0]

   Read-only properties:
                    Role: "peripheral"
        ConnectionConfig: [1x1 bluetoothLEConnectionConfig]
          TransmitBuffer: [1x1 struct]
                      ID: 2
```

Create a default Bluetooth LE configuration object to share the LL connection between the Central and Peripheral nodes.

```
cfgConnection = bluetoothLEConnectionConfig;
```

Specify the connection interval and connection offset. Throughout the simulation, the object establishes LL connection events for the duration of each connection interval. The connection offset is from the beginning of the connection interval.

```
cfgConnection.ConnectionInterval = 0.01; % In seconds
cfgConnection.ConnectionOffset = 0;      % In seconds
```

Specify the active communication period after the connection event is established between the Central and Peripheral nodes.

```
cfgConnection.ActivePeriod = 0.01 % In seconds
```

```
cfgConnection =
  bluetoothLEConnectionConfig with properties:

    ConnectionInterval: 0.0100
         AccessAddress: "5DA44270"
          UsedChannels: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... ]
             Algorithm: 1
          HopIncrement: 5
     CRCInitialization: "012345"
    SupervisionTimeout: 1
               PHYMode: "LE1M"
         InstantOffset: 6
      ConnectionOffset: 0
          ActivePeriod: 0.0100
```

Configure the connection between Central and Peripheral nodes by using the `configureConnection` object function of the `bluetoothLEConnectionConfig` object.

```
configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Specify the data rate in kb/s and the packet size in bytes. Enable packet generation to generate an application packet with a payload.

```
traffic = networkTrafficOnOff(DataRate=100, ...
                              PacketSize=10, ...
                              GeneratePacket=true);
```

Add application traffic from the Central to the Peripheral node by using the `addTrafficSource` object function.

```
addTrafficSource(centralNode,traffic,"DestinationNode",peripheralNode.Name);
```

Create a Bluetooth LE network consisting of a Central and a Peripheral node.

```
nodes = {centralNode peripheralNode};
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,nodes)
```

Set the simulation time in seconds and run the simulation.

```
simulationTime = 0.5;
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve application, link layer (LL), and physical layer (PHY) statistics corresponding to the broadcaster and receiver nodes. For more information about the statistics, see “Bluetooth LE Node Statistics”.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
    Name: "CentralNode"
      ID: 1
     App: [1x1 struct]
      LL: [1x1 struct]
     PHY: [1x1 struct]
```

```
peripheralStats = statistics(peripheralNode)
```

```
peripheralStats = struct with fields:
    Name: "PeripheralNode"
      ID: 2
     App: [1x1 struct]
      LL: [1x1 struct]
     PHY: [1x1 struct]
```

Input Arguments

`bluetoothLENodeObj` — Bluetooth LE node object

`bluetoothLENode` object

Bluetooth LE node object, specified as a `bluetoothLENode` object.

Output Arguments

`nodeStatistics` — Statistics of the Bluetooth LE node

structure

Statistics of the Bluetooth LE node, returned as a structure. For more information about this output, see “Bluetooth LE Node Statistics”.

Data Types: `struct`

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 12, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification". v5.3. https://www.bluetooth.com/.

See Also

Objects

`bluetoothLENode`

Functions

`addTrafficSource` | `channelInvokeDecision` | `pushChannelData` | `runNode` | `updateChannelList`

Topics

“Bluetooth LE Node Statistics”

updateChannelList

Provide updated channel list to Bluetooth LE node

Syntax

```
status = updateChannelList(bluetoothLENodeObj,newUsedChannelsList)
status = updateChannelList(bluetoothLENodeObj,newUsedChannelsList,
DestinationNode=destinationNode)
```

Description

`status = updateChannelList(bluetoothLENodeObj,newUsedChannelsList)` updates the channel map by providing a new list of used channels, `newUsedChannelsList`, to the Bluetooth low energy (LE) node specified by the `bluetoothLENodeObj` object. The object function returns the status, `status`, indicating whether the node accepted the new channel list. To use this syntax, set the Role property of `bluetoothLENode` to `"isochronous-broadcaster"`.

`status = updateChannelList(bluetoothLENodeObj,newUsedChannelsList, DestinationNode=destinationNode)` additionally specifies the destination node, `DestinationNode`, to which this object function provides a new list of used channels. To use this syntax, set the Role property of the `bluetoothLENode` object to `"central"` and the Role property of the destination node to `"peripheral"`.

Examples

Classify Channels and Update Channel Map in Bluetooth LE Network

Create a wireless network simulator.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create a Bluetooth LE node, specifying the role as `"central"`. Specify the position of the node.

```
centralNode = bluetoothLENode("central");
centralNode.Position = [0 0 0]                  % In x-, y-, z-direction, in meters
```

```
centralNode =
  bluetoothLENode with properties:

        TransmitterPower: 20
         TransmitterGain: 0
           ReceiverRange: 100
            ReceiverGain: 0
     ReceiverSensitivity: -100
             NoiseFigure: 0
    InterferenceFidelity: 0
                    Name: "Node1"
                Position: [0 0 0]

   Read-only properties:
```

```
                  Role: "central"
        ConnectionConfig: [1×1 bluetoothLEConnectionConfig]
         PeripheralCount: 0
          TransmitBuffer: [1×1 struct]
                      ID: 1
```

Create a Bluetooth LE node, specifying the role as `"peripheral"`. Specify the position of the node.

```
peripheralNode = bluetoothLENode("peripheral");
peripheralNode.Position = [10 0 0]              % In x-, y-, z-direction, in meters
```

```
peripheralNode =
  bluetoothLENode with properties:

        TransmitterPower: 20
         TransmitterGain: 0
           ReceiverRange: 100
            ReceiverGain: 0
     ReceiverSensitivity: -100
             NoiseFigure: 0
    InterferenceFidelity: 0
                    Name: "Node2"
                Position: [10 0 0]

   Read-only properties:
                    Role: "peripheral"
        ConnectionConfig: [1×1 bluetoothLEConnectionConfig]
          TransmitBuffer: [1×1 struct]
                      ID: 2
```

Create a Bluetooth LE configuration object, specifying the connection interval and active period for the connection.

```
cfgConnection = bluetoothLEConnectionConfig (...
    ConnectionInterval=0.02, ...                % In seconds
    ActivePeriod=0.02, ...                      % In seconds
    ConnectionOffset=0);                        % In seconds
```

Configure the link layer (LL) connection between the Central and Peripheral nodes.

```
configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create a `networkTrafficOnOff` object to generate an On-Off application traffic pattern. Specify the data rate in kb/s and the packet size in bytes. Enable packet generation to generate an application packet with a payload. Add application traffic from the Central to the Peripheral node by using the `addTrafficSource` object function.

```
traffic = networkTrafficOnOff(DataRate=500, ...
    PacketSize=10, ...
    GeneratePacket=true);
addTrafficSource(centralNode,traffic,"DestinationNode",peripheralNode.Name);
```

Create a WLAN node to introduce interference in the network by using the `helperInterferingWLANNode` helper object. Specify the position of the WLAN node. Specify the transmitter power and frequency of operation of the WLAN node. Specify the signal periodicity in which the WLAN node transmits the signal.

```
wlanNode = helperInterferingWLANNode( ...
Position=[2 0 0], ...                       % In x-, y-, z-direction, in meters
TransmitterPower=30, ...                    % In dBm
CenterFrequency=2.472e9, ...                % In Hz
SignalPeriodicity=2e-3);                    % In seconds
```

Create a Bluetooth LE network consisting of a Central node, a Peripheral node, and a WLAN node.

```
nodes = {centralNode peripheralNode wlanNode};
```

Add a listener for an event by using the `addlistener` function. This function attaches the listener to the source of the event, which is a handle object. When an event is triggered, the `addlistener` function invokes the callback function.

Assume that when the number of retransmitted packets in the Bluetooth LE network is greater than 100 and when the packet loss ratio is greater than 0.2, the used (good) channels list is updated to use only channels 1-10. This function is implemented in the callback function channelMapUpdateFcn on page 3-71. To check statistics and update the channels at the end of each connection event, create a listener at the Central node for the `ConnectionEventEnded` event. Because the Central node classifies and updates the channel map for all of the Peripheral nodes, a listener is created at the Central node.

```
addlistener(centralNode,"ConnectionEventEnded",@(nodeobj,eventdata) channelMapUpdateFcn(nodeobj,p
```

Create a function handle to display the updated list of used channels for Bluetooth LE communication.

```
channelMapDispFcnhandle = @(nodeobj,eventdata) disp(['Channel map is updated and the used channe
    num2str(eventdata.Data.UpdatedChannelList) ' at ' num2str(eventdata.Data.CurrentTime) ' secon
```

To display the updated list of channels after the Central node updates the channel map, create a listener at the Central node for the `ChannelMapUpdated` event. The listener returns the source node object and the event data. The event data is a structure containing the event information that triggered the listener.

```
addlistener(centralNode,"ChannelMapUpdated",channelMapDispFcnhandle);
```

Add the Central, Peripheral, and WLAN nodes to the wireless network simulator.

```
addNodes(networkSimulator,nodes)
```

Set the simulation time and run the simulation.

```
simulationTime = 0.3;                  % In seconds
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve application, link layer (LL) , and physical layer (PHY) statistics related to the source, relay, and destination nodes by using the `statistics` object function. For more information about the statistics, see “Bluetooth LE Node Statistics”.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
    Name: "Node1"
      ID: 1
     App: [1×1 struct]
```

```
      LL: [1×1 struct]
     PHY: [1×1 struct]
```

```
peripheralStats = statistics(peripheralNode)
```

```
peripheralStats = struct with fields:
    Name: "Node2"
      ID: 2
     App: [1×1 struct]
      LL: [1×1 struct]
     PHY: [1×1 struct]
```

The callback function of the `ConnectionEventEnded` event channelMapUpdateFcn on page 3-71 retrieves the statistics corresponding to the Bluetooth LE node object. This function checks whether these conditions are true.

- The number of retransmitted packets in the Bluetooth LE network is greater than 100.
- The packet loss ratio is greater than 0.2.

If the conditions are true, the function updates the channel map in the Bluetooth LE node object to use Bluetooth LE channels 1-10 by using the `updateChannelList` object function of the `bluetoothLENode` object.

```
function channelMapUpdateFcn(nodeobj,peripheralName)
usedChannels = 1:10;
stats = statistics(nodeobj);
if stats.LL.RetransmittedDataPackets > 100 && stats.LL.PacketLossRatio > 0.2
    updateChannelList(nodeobj,usedChannels,"DestinationNode",peripheralName);
end
end
```

Input Arguments

`bluetoothLENodeObj` — Bluetooth LE node object

`bluetoothLENode` object

Bluetooth LE node object, specified as a `bluetoothLENode` object.

`newUsedChannelsList` — List of good channels

integer vector with element values in the range [0, 36]

List of good channels, specified as an integer vector with element values in the range [0, 36]. This input specifies the new list of used channels to update the channel map for the specified Bluetooth LE node. To ensure that at least two channels are set as used (good) channels, specify a vector length greater than 1.

`DestinationNode` — Name of destination node

character vector | string scalar

Name of destination node, specified as a character vector or string scalar. If you set the Role property of the `bluetoothLENodeObj` object to `"central"` or `"peripheral"`, this input is mandatory. This

input specifies the Name property of the specified Bluetooth LE node object, bluetoothLENodeObj, for sending data to the specified destination node.

Data Types: `char` | `string`

Output Arguments

`status` — Flag indicating whether LL accepts the new channel list
`0` or `false` | `1` or `true`

Flag indicating whether LL of the specified Bluetooth LE node `bluetoothLENodeObj` accepts the new channel list, returned as `0` (`false`) or `1` (`true`).

Data Types: `logical`

Version History

Introduced in R2022a

References

[1] Bluetooth Technology Website. “Bluetooth Technology Website | The Official Website of Bluetooth Technology.” Accessed November 12, 2021. https://www.bluetooth.com/.

[2] Bluetooth Special Interest Group (SIG). "Bluetooth Core Specification". v5.3. https://www.bluetooth.com/.

See Also

Objects
`bluetoothLENode`

Functions
`addTrafficSource` | `channelInvokeDecision` | `pushChannelData` | `runNode` | `statistics`

writeCustomBlock

Write custom block to PCAPNG file

Syntax

`writeCustomBlock(pcapngObj,customData)`

Description

`writeCustomBlock(pcapngObj,customData)` writes the custom block data, `customData`, to a PCAPNG file specified in the PCAPNG file writer object.

Examples

Write Bluetooth LE User-Defined Custom Block to PCAPNG File

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapngObj = pcapngWriter('FileName','writeBluetoothLEcustomdata');
```

Write the interface description block for Bluetooth low energy (LE).

```
interfaceName = 'Bluetooth LE interface';
bleLinkType = 251;
interfaceId = writeInterfaceDescriptionBlock(pcapngObj,bleLinkType, ...
    interfaceName);
```

Write the custom block to specify user-defined data.

```
writeCustomBlock(pcapngObj,"This block writes user-defined data");
```

Input Arguments

Note The `pcapngWriter` object does not overwrite the existing PCAPNG file. Each time when you create this object, specify a unique PCAPNG file name.

`pcapngObj` — PCAPNG file writer object

`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`customData` — User-defined data

character vector | string scalar

User-defined data, specified as a character vector or a string scalar.

Data Types: `char` | `string`

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`write` | `writeInterfaceDescriptionBlock`

Objects

`pcapWriter` | `pcapngWriter`

writeGlobalHeader

Write global header to PCAP file

Syntax

```
writeGlobalHeader(pcapObj,linkType)
```

Description

`writeGlobalHeader(pcapObj,linkType)` writes a global header to the PCAP file specified in the PCAP file writer object, `pcapObj`. Input `linkType` specifies the unique identifier for a protocol.

Examples

Write Bluetooth LE Packet Global Header to PCAP File

Create a PCAP file writer object, specifying the name of the PCAP file.

```
pcapObj = pcapWriter('FileName','writeBluetoothLEheader');
```

Specify the Bluetooth low energy (LE) link type.

```
bleLinkType = 251;
```

Write a global header to the PCAP file.

```
writeGlobalHeader(pcapObj,bleLinkType);
```

Input Arguments

Note The `pcapWriter` object does not overwrite the existing PCAP file. Each time when you create this object, specify a unique PCAP file name.

`pcapObj` — PCAP file writer object

`pcapWriter` object

PCAP file writer object, specified as a `pcapWriter` object.

`linkType` — Unique identifier for a protocol

integer in the range [0, 65,535].

Unique identifier for a protocol, specified as an integer in the range [0, 65535].

Data Types: `double`

Version History

Introduced in R2020b

References

[1] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[2] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`write`

Objects

`pcapWriter` | `pcapngWriter`

writeInterfaceDescriptionBlock

Write interface description block to PCAPNG file

Syntax

```
interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface)
```

Description

`interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface)` writes an interface description block to the PCAPNG file specified in the PCAPNG file writer object, `pcapngObj`. Input `linkType` specifies the unique identifier for the protocol and input `interface` specifies the interface on which the protocol packets are captured. This object function returns the unique identifier for the interface.

Examples

Write Bluetooth LE Interface Description Block to PCAPNG File

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapngObj = pcapngWriter('FileName','writeBluetoothLEinterface');
```

Write the interface description block for Bluetooth low energy (LE).

```
interfaceName = 'Bluetooth LE interface';
bleLinkType = 251;
interfaceId = writeInterfaceDescriptionBlock(pcapngObj,bleLinkType, ...
    interfaceName);
```

Input Arguments

Note The `pcapngWriter` object does not overwrite the existing PCAPNG file. Each time when you create this object, specify a unique PCAPNG file name.

`pcapngObj` — PCAPNG file writer object

`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`linkType` — Unique identifier for protocol

integer in the range [0, 65,535].

Unique identifier for a protocol, specified as an integer in the range [0, 65,535].

Data Types: `double`

`interface` — Name of the interface on which protocol packets are captured
character vector | string scalar

Name of the interface on which protocol packets are captured, specified as a character vector or a string scalar in 8-bit unicode transformation format (UTF-8) format.

Data Types: `char` | `string`

Output Arguments

`interfaceID` — Unique identifier for an interface
nonnegative scalar

Unique identifier for an interface, specified as a nonnegative scalar.

Data Types: `double`

Version History

Introduced in R2020b

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org.

Extended Capabilities

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

See Also

Functions
`write` | `writeCustomBlock`

Objects
`pcapWriter` | `pcapngWriter`

addChannelModel

Add custom channel or path loss model

Syntax

```
addChannelModel(networkSimulator,customMdl)
```

Description

`addChannelModel(networkSimulator,customMdl)` adds a custom channel model or path loss model to the wireless network simulation. The `addChannelModel` function sets the `ChannelFunction` property of the `wirelessNetworkSimulator` object to the custom model specified at input.

Input Arguments

`networkSimulator` — Wireless network simulator

`wirelessNetworkSimulator` object

Wireless network simulator, specified as a `wirelessNetworkSimulator` object.

`customMdl` — Function for computing custom channel or path loss model

MATLAB function handle

Function for computing the custom channel or path loss model, specified as a MATLAB function handle. The syntax for the custom function must be of the format:

```
rxData = customFcnName(rxInfo,txData)
```

The `rxInfo` input is the receiver node information, and the `txData` input specifies the transmitted packets. The simulator automatically passes information about the receiver node and the packets transmitted by a transmitter node as inputs to the custom function.

The receiver node information is a structure with these fields.

Field	Description
`ID`	Unique receiver node identifier.
`Position`	Position of the receiver node in 3-D Cartesian coordinates (`x`, `y`, `z`). Units are in meters.
`Velocity`	Velocity of the receiver node (V_{rx}, V_{ry}, V_{rz}) in the `x`-, `y`-, and `z`-directions. Units are in meters per second.

The packets from the transmitter node is a structure with these fields.

Field	Description

`Type`	Type of input signal packet, specified as one of these values: 0, 1, 2, 3, and 4. • 0 — Invalid packet • 1 — WLAN packet • 2 — 5G packet • 3 — Bluetooth LE packet • 4 — Bluetooth BR/EDR packet
`TransmitterID`	Unique transmitter node identifier, specified as a positive scalar integer.
`TransmitterPosition`	Position of the transmitter node in 3-D Cartesian coordinates (`x`, `y`, `z`). Units are in meters.
`TransmitterVelocity`	Velocity of the transmitter node (V_{tx}, V_{ty}, V_{tz}) in the `x`-, `y`-, and `z`-directions. Units are in meters per second.
`StartTime`	Time at which the transmitter starts transmitting the packets, specified as a non-negative scalar. Units are in seconds.
`Duration`	Duration of the transmitter packet, specified as a positive scalar. Units are in seconds.
`Power`	Average power consumption during transmission. Units are in dBm.
`CenterFrequency`	Center frequency of the carrier signal. Units are in Hz.
`Bandwidth`	Carrier signal bandwidth. Units are in Hz.
`Abstraction`	Type of abstraction, specified as a logical scalar. • 1 — Abstracted physical layer • 0 — Full physical layer The default value is 0.
`SampleRate`	Sample rate of the packet, specified as a numeric scalar in samples per second. This field is applicable only if the value of the `Abstraction` field is set to 0. The default value is [ ].
`DirectToDestination`	Information about the transmitted packet, specified as a numeric scalar integer. • The value is 0 if a normal packet is transmitted over the channel. • The value is a non-zero scalar integer if the packet bypasses the channel model and is transmitted directly to the destination node. In this case, this field value specifies the destination node identifier.

`Data`	Time-domain samples or frame information. • The field contains time-domain samples of the transmitted packet for full physical layer. In this case, the value of the `Abstraction` field is 0. • The field contains frame information for the abstracted physical layer. In this case, the value of the `Abstraction` field is 1.
`Metadata`	A structure representing the technology-specific and abstraction-specific information of the packet.

The custom channel model function applies channel effects to the packets transmitted by the simulator. The modified packets are then sent back to the must simulator. The packets returned at the output must be a structure with same fields as those of the transmitted packets.

Data Types: `function_handle`

Version History

Introduced in R2022b

See Also

Objects

`wirelessNetworkSimulator` | `bluetoothLENode` | `bluetoothNode`

Functions

`addNodes` | `scheduleAction` | `cancelAction` | `run`

Topics

“Create, Configure, and Simulate Bluetooth BR Piconet”
“Bluetooth BR Data and Voice Communication with WLAN Signal Interference”
“Simulate Multiple Bluetooth BR Piconets with ACL Traffic”

addNodes

Add nodes to simulation

Syntax

`addNodes(networkSimulator,nodes)`

Description

`addNodes(networkSimulator,nodes)` adds the specified wireless nodes to the `wirelessNetworkSimulator` object. You must add the nodes to the simulator before running the simulation.

Examples

Simulate Bluetooth BR Network with Default Channel Effects

Create a `wirelessNetworkSimulator` object. By default, the `wirelessNetworkSimulator` object applies free-space path loss model for the channel effects.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and other with the `"peripheral"` role. Specify the position of the Peripheral node in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
          GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Specify the simulation time in seconds.

```
simulationTime = 0.05;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve application, baseband, and physical layer (PHY) statistics corresponding to the Central and Peripheral nodes.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
        Name: "Node1"
          ID: 1
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]


peripheralStats = statistics(peripheralNode)

peripheralStats = struct with fields:
        Name: "Node2"
          ID: 2
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

Input Arguments

`networkSimulator` — Wireless network simulator

`wirelessNetworkSimulator` object

Wireless network simulator, specified as a `wirelessNetworkSimulator` object.

`nodes` — Nodes to transmit and receive data

`bluetoothLENode` object | `bluetoothNode` object

Nodes to transmit and receive data, specified as a `bluetoothLENode` object or `bluetoothNode` object.

Version History

Introduced in R2022b

See Also

Objects

`wirelessNetworkSimulator` | `bluetoothLENode` | `bluetoothNode`

Functions

`run` | `addChannelModel` | `scheduleAction` | `cancelAction`

Topics
“Create, Configure, and Simulate Bluetooth BR Piconet”
“Bluetooth BR Data and Voice Communication with WLAN Signal Interference”
“Simulate Multiple Bluetooth BR Piconets with ACL Traffic”

cancelAction

Cancel scheduled action

Syntax

```
cancelAction(networkSimulator,actionID)
```

Description

`cancelAction(networkSimulator,actionID)` cancels an action scheduled to be performed during the specified wireless network simulation.

Examples

Schedule and Cancel Actions During Network Simulation

Simulate Bluetooth BR Network

Create a `wirelessNetworkSimulator` object. By default, the `wirelessNetworkSimulator` object applies free-space path loss model for the channel effects.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and other with the `"peripheral"` role. Specify the position of the Peripheral node in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
    GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Schedule Action

Create a custom function `statusInfo` to define an one time action and periodic action for the simulator to perform during simulation. The input data for the custom function is stored as a structure and passed using the `scheduleAction` function. For the one time action, the `statusInfo` function displays the node details when the simulation time is 0.001s. For the periodic action, the `statusInfo` function displays the physical layer statistics of the central node in the Bluetooth BR network.

Specify the `ActionType` input argument of `statusInfo` function as `"OneTime"`. Also, specify the names of the nodes in the network as input.

```
userdata = struct(ActionType="OneTime",Nodes=[centralNode peripheralNode]);
```

Configure the simulator by using the `scheduleAction` function to display the node details when the simulation time is 0.001s.

```
startTime = 0.001;
actionID1 = scheduleAction(networkSimulator,@statusInfo,userdata,startTime);
```

Specify the `ActionType` input argument value of `statusInfo` function as `"Periodic"`. Also, specify the central node and the simulator as inputs.

```
userdata = struct(ActionType="Periodic",CentralNode=centralNode,Simulator=networkSimulator);
```

Configure the simulator by using the `scheduleAction` function to display the physical layer statistics of the central node at a time interval of 0.02s starting from the beginning of the simulation.

```
startTime = 0;
periodicity = 0.02;
actionID2 = scheduleAction(networkSimulator,@statusInfo,userdata,startTime,periodicity);
```

Cancel Action

To cancel a scheduled action, use the `cancelAction` function. Specify the value of the unique identifier denoting the action. The identifier is returned by the `scheduleAction` function.

```
cancelAction(networkSimulator,actionID1)
```

Run Simulation

Specify the simulation time in seconds.

```
simulationTime = 0.1;
```

Run the simulation for the specified simulation time. You can notice that the one time action scheduled to perform at 0.001s is cancelled. The simulator performs only the scheduled periodic action.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
-------Periodic Action -------
Statistics of Node1 at time 0.000 seconds:
                   ReceivedPackets: 0
                    DecodeFailures: 0
                  PacketCollisions: 0
               CoChannelCollisions: 0
```

```
              CollisionsWithBREDR: 0
           CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
               TransmittedPackets: 1
                  TransmittedBits: 366

-------End-------
-------Periodic Action -------
Statistics of Node1 at time 0.020 seconds:
                   ReceivedPackets: 16
                    DecodeFailures: 0
                  PacketCollisions: 0
               CoChannelCollisions: 0
               CollisionsWithBREDR: 0
            CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
               TransmittedPackets: 17
                  TransmittedBits: 6222

-------End-------
-------Periodic Action -------
Statistics of Node1 at time 0.040 seconds:
                   ReceivedPackets: 32
                    DecodeFailures: 0
                  PacketCollisions: 0
               CoChannelCollisions: 0
               CollisionsWithBREDR: 0
            CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
               TransmittedPackets: 33
                  TransmittedBits: 12078

-------End-------
-------Periodic Action -------
Statistics of Node1 at time 0.060 seconds:
                   ReceivedPackets: 48
                    DecodeFailures: 0
                  PacketCollisions: 0
               CoChannelCollisions: 0
               CollisionsWithBREDR: 0
            CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
               TransmittedPackets: 49
                  TransmittedBits: 17934

-------End-------
-------Periodic Action -------
Statistics of Node1 at time 0.080 seconds:
                   ReceivedPackets: 64
                    DecodeFailures: 0
                  PacketCollisions: 0
               CoChannelCollisions: 0
               CollisionsWithBREDR: 0
            CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
               TransmittedPackets: 65
                  TransmittedBits: 23790
```

```
-------End-------
-------Periodic Action -------
Statistics of Node1 at time 0.100 seconds:
                   ReceivedPackets: 80
                    DecodeFailures: 0
                  PacketCollisions: 0
               CoChannelCollisions: 0
               CollisionsWithBREDR: 0
            CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
                TransmittedPackets: 81
                   TransmittedBits: 29646

-------End-------

function statusInfo(~,userdata)
switch(userdata.ActionType)
    case "Periodic"
        fprintf("-------Periodic Action -------\n")
        fprintf("Statistics of %s at time %.3f seconds:\n",userdata.CentralNode.Name,userdata.Si
        stats = statistics(userdata.CentralNode);
        disp(stats.PHY)
        fprintf("-------End-------\n")
    case "OneTime"
        fprintf("-------One Time Action -------\n")
        fprintf("Node details:\n")
        for idx=1:numel(userdata.Nodes)
            fprintf("Name: %s ID: %d Role: %s\n",userdata.Nodes(idx).Name,userdata.Nodes(idx).ID
        end
        fprintf("-------End-------\n")
end
end
```

Input Arguments

`networkSimulator` — Wireless network simulator

`wirelessNetworkSimulator` object

Wireless network simulator, specified as a `wirelessNetworkSimulator` object.

`actionID` — Unique identifier of action to cancel

positive scalar integer

Unique identifier of action to cancel, specified as a positive scalar integer. The unique identifier is returned by the `scheduleAction` function.

Version History

Introduced in R2022b

See Also

Objects

`wirelessNetworkSimulator` | `bluetoothLENode` | `bluetoothNode`

Functions
`addChannelModel` | `scheduleAction` | `addNodes` | `run`

Topics
“Create, Configure, and Simulate Bluetooth BR Piconet”
“Bluetooth BR Data and Voice Communication with WLAN Signal Interference”
“Simulate Multiple Bluetooth BR Piconets with ACL Traffic”

run

Run simulation

Syntax

```
run(networkSimulator,simDuration)
```

Description

`run(networkSimulator,simDuration)` runs the wireless network simulation for the specified simulation time duration. If any action is scheduled at a particular simulation time, the `run` function performs the scheduled action. Use the `scheduleAction` function to schedule an action to be performed during the simulation process.

You must call the `run` function only once after initializing the `wirelessNetworkSimulator` object.

Examples

Simulate Bluetooth BR Network with Default Channel Effects

Create a `wirelessNetworkSimulator` object. By default, the `wirelessNetworkSimulator` object applies free-space path loss model for the channel effects.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and other with the `"peripheral"` role. Specify the position of the Peripheral node in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
          GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Specify the simulation time in seconds.

```
simulationTime = 0.05;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```

Retrieve application, baseband, and physical layer (PHY) statistics corresponding to the Central and Peripheral nodes.

```
centralStats = statistics(centralNode)
```

```
centralStats = struct with fields:
        Name: "Node1"
          ID: 1
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

```
peripheralStats = statistics(peripheralNode)
```

```
peripheralStats = struct with fields:
        Name: "Node2"
          ID: 2
         App: [1x1 struct]
    Baseband: [1x1 struct]
         PHY: [1x1 struct]
```

Schedule Action to Perform During Network Simulation

Simulate Bluetooth BR Network

Create a `wirelessNetworkSimulator` object. By default, the `wirelessNetworkSimulator` object applies free-space path loss model for the channel effects.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and other with the `"peripheral"` role. Specify the position of the Peripheral node in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
    GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Schedule Action

Create a custom function `displayInfo` to define the actions for the simulator to perform during simulation. You can define an one time action, periodic action, and action when simulator advances in time. The input data for the custom function is stored as a structure and passed using the `scheduleAction` function.

A) Schedule One Time Action

For the one time action, the `displayInfo` function displays the details of the nodes in the network.

Specify the `ActionType` input argument value of `displayInfo` function as `"OneTime"`. Specify the names of the nodes in the network as input to `displayInfo` function.

```
userdata = struct(ActionType="OneTime",Nodes=[centralNode peripheralNode]);
```

Configure the simulator by using the `scheduleAction` function to display the node details when the simulation time is 0.001s.

```
startTime = 0.001;
scheduleAction(networkSimulator,@displayInfo,userdata,startTime);
```

B) Schedule Periodic Action

For the periodic action, the `displayInfo` function displays the physical layer statistics of the central node in the Bluetooth BR network.

Specify the `ActionType` input argument value of `displayInfo` function as `"Periodic"`. Specify the central node and the simulator as inputs to `displayInfo` function.

```
userdata = struct(ActionType="Periodic",CentralNode=centralNode,Simulator=networkSimulator);
```

Configure the simulator by using the `scheduleAction` function to display the physical layer statistics of the central node at a time interval of 0.02s starting from the beginning of the simulation.

```
startTime = 0;
periodicity = 0.02;
scheduleAction(networkSimulator,@displayInfo,userdata,startTime,periodicity);
```

C) Schedule Action When Simulator Advances in Time

When the simulator advances in time, the `displayInfo` function displays the next event time of the simulator.

Specify the `ActionType` input argument value of `displayInfo` function as `"TimeAdvance"`. Specify the simulator as input to the `displayInfo` function.

```
userdata = struct(ActionType="TimeAdvance",Simulator=networkSimulator);
```

Set the periodicity value to 0. Configure the simulator by using the `scheduleAction` function to display the time of the next event.

```
startTime = 0;
periodicity = 0;
scheduleAction(networkSimulator,@displayInfo,userdata,startTime,periodicity);
```

Run Simulation

Specify the simulation time in seconds.

```
simulationTime = 0.05;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
-------Periodic Action -------
Statistics of Node1 at time 0.000 seconds:
                   ReceivedPackets: 0
                    DecodeFailures: 0
                  PacketCollisions: 0
               CoChannelCollisions: 0
               CollisionsWithBREDR: 0
            CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
                TransmittedPackets: 1
                   TransmittedBits: 366

-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00037 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00063 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00075 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00100 seconds
-------End-------
-------One Time Action -------
Node details:
Name: Node1 ID: 1 Role: central
Name: Node2 ID: 2 Role: peripheral
-------End-------
```

```
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00108 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00125 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00162 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00187 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00200 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00216 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00250 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00287 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00313 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00324 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00325 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00375 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00412 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00432 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00438 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00450 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00500 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00537 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00540 seconds
-------End-------
-------Action When Simulator Advances in Time-------
```

```
Simulator next event time is at 0.00562 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00575 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00625 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00648 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00662 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00688 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00750 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00756 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00787 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00813 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00825 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00864 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00875 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00912 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00937 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00950 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00972 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01000 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01037 seconds
```

```
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01063 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01075 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01080 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01125 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01162 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01188 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01188 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01200 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01250 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01287 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01296 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01312 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01325 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01375 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01404 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01412 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01438 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01450 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01500 seconds
-------End-------
```

```
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01512 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01537 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01562 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01575 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01620 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01625 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01662 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01688 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01728 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01750 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01787 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01812 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01825 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01836 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01875 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01912 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01937 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01944 seconds
-------End-------
-------Action When Simulator Advances in Time-------
```

```
Simulator next event time is at 0.01950 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02000 seconds
-------End-------
-------Periodic Action -------
Statistics of Node1 at time 0.020 seconds:
                   ReceivedPackets: 16
                    DecodeFailures: 0
                  PacketCollisions: 0
               CoChannelCollisions: 0
               CollisionsWithBREDR: 0
            CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
                TransmittedPackets: 17
                   TransmittedBits: 6222

-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02037 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02052 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02063 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02075 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02125 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02160 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02162 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02187 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02200 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02250 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02268 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02287 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02312 seconds
-------End-------
-------Action When Simulator Advances in Time-------
```

```
Simulator next event time is at 0.02325 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02375 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02376 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02412 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02438 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02450 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02484 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02500 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02537 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02562 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02575 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02592 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02625 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02662 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02687 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02750 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02787 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02808 seconds
```

```
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02813 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02825 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02875 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02912 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02916 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02937 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02950 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03000 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03024 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03037 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03062 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03075 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03125 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03132 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03162 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03188 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03200 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03240 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03250 seconds
-------End-------
```

```
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03287 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03313 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03325 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03348 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03375 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03412 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03438 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03450 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03456 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03500 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03537 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03562 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03564 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03575 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03625 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03662 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03672 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03687 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
```

```
Simulator next event time is at 0.03750 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03780 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03787 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03812 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03825 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03875 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03888 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03912 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03938 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03950 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03996 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04000 seconds
-------End-------
-------Periodic Action -------
Statistics of Node1 at time 0.040 seconds:
                   ReceivedPackets: 32
                    DecodeFailures: 0
                  PacketCollisions: 0
               CoChannelCollisions: 0
               CollisionsWithBREDR: 0
            CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
                TransmittedPackets: 33
                   TransmittedBits: 12078

-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04037 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04063 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04075 seconds
-------End-------
-------Action When Simulator Advances in Time-------
```

```
Simulator next event time is at 0.04104 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04125 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04162 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04188 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04200 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04212 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04250 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04287 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04312 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04320 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04325 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04375 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04412 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04428 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04437 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04450 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04500 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04536 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04537 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04562 seconds
```

```
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04575 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04625 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04644 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04662 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04688 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04750 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04752 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04787 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04813 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04825 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04860 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04875 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04912 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04938 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04950 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04968 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.05000 seconds
-------End-------

function displayInfo(~,userdata)
switch(userdata.ActionType)
```

```
    case "Periodic"
        fprintf("-------Periodic Action -------\n")
        fprintf("Statistics of %s at time %.3f seconds:\n",userdata.CentralNode.Name,userdata.Si
        stats = statistics(userdata.CentralNode);
        disp(stats.PHY)
        fprintf("-------End-------\n")
    case "OneTime"
        fprintf("-------One Time Action -------\n")
        fprintf("Node details:\n")
        for idx=1:numel(userdata.Nodes)
            fprintf("Name: %s ID: %d Role: %s\n",userdata.Nodes(idx).Name,userdata.Nodes(idx).ID
        end
        fprintf("-------End-------\n")
    case "TimeAdvance"
        fprintf("-------Action When Simulator Advances in Time-------\n")
        fprintf("Simulator next event time is at %.5f seconds\n",userdata.Simulator.CurrentTime)
        fprintf("-------End-------\n")
end
end
```

Input Arguments

`networkSimulator` — Wireless network simulator

`wirelessNetworkSimulator` object

Wireless network simulator, specified as a `wirelessNetworkSimulator` object.

`simDuration` — Duration of simulation

positive scalar

Duration of simulation, specified as a positive scalar. Units are in seconds.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Version History

Introduced in R2022b

See Also

Objects

`wirelessNetworkSimulator` | `bluetoothLENode` | `bluetoothNode`

Functions

`addChannelModel` | `scheduleAction` | `cancelAction` | `addNodes`

Topics

“Create, Configure, and Simulate Bluetooth BR Piconet”
“Bluetooth BR Data and Voice Communication with WLAN Signal Interference”
“Simulate Multiple Bluetooth BR Piconets with ACL Traffic”

scheduleAction

Schedule action to perform during simulation

Syntax

```
scheduleAction(networkSimulator,callbackFcn,userData,startTime)
scheduleAction(___,periodicity)
actionID = scheduleAction(___)
```

Description

`scheduleAction(networkSimulator,callbackFcn,userData,startTime)` schedules an action to perform during the wireless network simulation. The `callbackFcn` argument specifies the action to be performed at the specified time `startTime`, and `userData` specifies the input data to use for the specified action.

`scheduleAction(___,periodicity)` specifies to repeat the action at regular or irregular time intervals, in addition to all input arguments from the previous syntax.

`actionID = scheduleAction(___)` returns an identifier specifying the action scheduled.

Examples

Schedule Action to Perform During Network Simulation

Simulate Bluetooth BR Network

Create a `wirelessNetworkSimulator` object. By default, the `wirelessNetworkSimulator` object applies free-space path loss model for the channel effects.

```
networkSimulator = wirelessNetworkSimulator.init();
```

Create two Bluetooth BR nodes, one with the `"central"` role and other with the `"peripheral"` role. Specify the position of the Peripheral node in meters.

```
centralNode = bluetoothNode("central");
peripheralNode = bluetoothNode("peripheral",Position=[1 0 0]);
```

Create a default Bluetooth BR connection configuration object to configure and share a connection between Bluetooth BR Central and Peripheral nodes.

```
cfgConnection = bluetoothConnectionConfig;
```

Configure connection between the Central and the Peripheral nodes.

```
connection = configureConnection(cfgConnection,centralNode,peripheralNode);
```

Create and configure a `networkTrafficOnOff` object to generate an On-Off application traffic pattern.

```
traffic = networkTrafficOnOff(DataRate=200,PacketSize=27, ...
    GeneratePacket=true,OnTime=inf);
```

Add application traffic from the Central to the Peripheral node.

```
addTrafficSource(centralNode,traffic,DestinationNode=peripheralNode);
```

Add the Central and Peripheral nodes to the wireless network simulator.

```
addNodes(networkSimulator,[centralNode peripheralNode]);
```

Schedule Action

Create a custom function `displayInfo` to define the actions for the simulator to perform during simulation. You can define an one time action, periodic action, and action when simulator advances in time. The input data for the custom function is stored as a structure and passed using the `scheduleAction` function.

A) Schedule One Time Action

For the one time action, the `displayInfo` function displays the details of the nodes in the network.

Specify the `ActionType` input argument value of `displayInfo` function as `"OneTime"`. Specify the names of the nodes in the network as input to `displayInfo` function.

```
userdata = struct(ActionType="OneTime",Nodes=[centralNode peripheralNode]);
```

Configure the simulator by using the `scheduleAction` function to display the node details when the simulation time is 0.001s.

```
startTime = 0.001;
scheduleAction(networkSimulator,@displayInfo,userdata,startTime);
```

B) Schedule Periodic Action

For the periodic action, the `displayInfo` function displays the physical layer statistics of the central node in the Bluetooth BR network.

Specify the `ActionType` input argument value of `displayInfo` function as `"Periodic"`. Specify the central node and the simulator as inputs to `displayInfo` function.

```
userdata = struct(ActionType="Periodic",CentralNode=centralNode,Simulator=networkSimulator);
```

Configure the simulator by using the `scheduleAction` function to display the physical layer statistics of the central node at a time interval of 0.02s starting from the beginning of the simulation.

```
startTime = 0;
periodicity = 0.02;
scheduleAction(networkSimulator,@displayInfo,userdata,startTime,periodicity);
```

C) Schedule Action When Simulator Advances in Time

When the simulator advances in time, the `displayInfo` function displays the next event time of the simulator.

Specify the `ActionType` input argument value of `displayInfo` function as `"TimeAdvance"`. Specify the simulator as input to the `displayInfo` function.

```
userdata = struct(ActionType="TimeAdvance",Simulator=networkSimulator);
```

Set the periodicity value to 0. Configure the simulator by using the `scheduleAction` function to display the time of the next event.

```
startTime = 0;
periodicity = 0;
scheduleAction(networkSimulator,@displayInfo,userdata,startTime,periodicity);
```

Run Simulation

Specify the simulation time in seconds.

```
simulationTime = 0.05;
```

Run the simulation for the specified simulation time.

```
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
-------Periodic Action -------
Statistics of Node1 at time 0.000 seconds:
                    ReceivedPackets: 0
                     DecodeFailures: 0
                   PacketCollisions: 0
                CoChannelCollisions: 0
                CollisionsWithBREDR: 0
             CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
                 TransmittedPackets: 1
                    TransmittedBits: 366

-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00037 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00063 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00075 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00100 seconds
-------End-------
-------One Time Action -------
Node details:
Name: Node1 ID: 1 Role: central
Name: Node2 ID: 2 Role: peripheral
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00108 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00125 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00162 seconds
-------End-------
```

```
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00187 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00200 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00216 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00250 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00287 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00313 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00324 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00325 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00375 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00412 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00432 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00438 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00450 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00500 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00537 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00540 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00562 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00575 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00625 seconds
-------End-------
-------Action When Simulator Advances in Time-------
```

```
Simulator next event time is at 0.00648 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00662 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00688 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00750 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00756 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00787 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00813 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00825 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00864 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00875 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00912 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00937 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00950 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.00972 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01000 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01037 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01063 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01075 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01080 seconds
```

```
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01125 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01162 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01188 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01188 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01200 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01250 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01287 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01296 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01312 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01325 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01375 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01404 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01412 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01438 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01450 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01500 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01512 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01537 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01562 seconds
-------End-------
```

```
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01575 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01620 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01625 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01662 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01688 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01728 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01750 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01787 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01812 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01825 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01836 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01875 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01912 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01937 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01944 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.01950 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02000 seconds
-------End-------
-------Periodic Action -------
Statistics of Node1 at time 0.020 seconds:
                   ReceivedPackets: 16
                    DecodeFailures: 0
```

```
                  PacketCollisions: 0
               CoChannelCollisions: 0
               CollisionsWithBREDR: 0
            CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
                TransmittedPackets: 17
                   TransmittedBits: 6222

-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02037 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02052 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02063 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02075 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02125 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02160 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02162 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02187 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02200 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02250 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02268 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02287 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02312 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02325 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02375 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02376 seconds
-------End-------
-------Action When Simulator Advances in Time-------
```

```
Simulator next event time is at 0.02412 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02438 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02450 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02484 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02500 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02537 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02562 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02575 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02592 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02625 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02662 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02687 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02750 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02787 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02808 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02813 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02825 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02875 seconds
```

```
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02912 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02916 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02937 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.02950 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03000 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03024 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03037 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03062 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03075 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03125 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03132 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03162 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03188 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03200 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03240 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03250 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03287 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03313 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03325 seconds
-------End-------
```

```
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03348 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03375 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03412 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03438 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03450 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03456 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03500 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03537 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03562 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03564 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03575 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03625 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03662 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03672 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03687 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03750 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03780 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03787 seconds
-------End-------
-------Action When Simulator Advances in Time-------
```

```
Simulator next event time is at 0.03812 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03825 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03875 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03888 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03912 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03938 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03950 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.03996 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04000 seconds
-------End-------
-------Periodic Action -------
Statistics of Node1 at time 0.040 seconds:
                   ReceivedPackets: 32
                    DecodeFailures: 0
                  PacketCollisions: 0
               CoChannelCollisions: 0
               CollisionsWithBREDR: 0
            CollisionsWithNonBREDR: 0
    CollisionsWithBREDRAndNonBREDR: 0
                TransmittedPackets: 33
                   TransmittedBits: 12078

-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04037 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04063 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04075 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04104 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04125 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04162 seconds
-------End-------
-------Action When Simulator Advances in Time-------
```

```
Simulator next event time is at 0.04188 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04200 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04212 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04250 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04287 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04312 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04320 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04325 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04375 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04412 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04428 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04437 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04450 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04500 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04536 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04537 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04562 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04575 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04625 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04644 seconds
```

```
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04662 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04688 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04700 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04750 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04752 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04787 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04813 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04825 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04860 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04875 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04912 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04938 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04950 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.04968 seconds
-------End-------
-------Action When Simulator Advances in Time-------
Simulator next event time is at 0.05000 seconds
-------End-------

function displayInfo(~,userdata)
switch(userdata.ActionType)
    case "Periodic"
        fprintf("-------Periodic Action -------\n")
        fprintf("Statistics of %s at time %.3f seconds:\n",userdata.CentralNode.Name,userdata.Si
        stats = statistics(userdata.CentralNode);
        disp(stats.PHY)
        fprintf("-------End-------\n")
    case "OneTime"
        fprintf("-------One Time Action -------\n")
        fprintf("Node details:\n")
```

```
        for idx=1:numel(userdata.Nodes)
            fprintf("Name: %s ID: %d Role: %s\n",userdata.Nodes(idx).Name,userdata.Nodes(idx).ID
        end
        fprintf("-------End-------\n")
    case "TimeAdvance"
        fprintf("-------Action When Simulator Advances in Time-------\n")
        fprintf("Simulator next event time is at %.5f seconds\n",userdata.Simulator.CurrentTime)
        fprintf("-------End-------\n")
end
end
```

Input Arguments

`networkSimulator` — Wireless network simulator

`wirelessNetworkSimulator` object

Wireless network simulator, specified as a `wirelessNetworkSimulator` object.

`callbackFcn` — Callback function to run at specified time

function handle

Callback function to run at the specified time, specified as a function handle. The callback function is a custom function defining an action to be performed during the wireless network simulation. The syntax for the callback function must be:

```
callbackFcn(actionID,userData)
```

`userData` — Input to callback function

`[ ]` | structure | cell array | numeric value | character vector | string scalar

Input to the callback function, specified as a numeric value, character vector, or string scalar. If the callback function does not require any input, specify the `userData` value as `[ ]`. To pass multiple values as input to the callback function, specify the input as a structure or a cell array.

`startTime` — Absolute simulation time to perform the action

non-negative scalar

Absolute simulation time at which a scheduled action is to be performed, specified as a non-negative scalar. Units are in seconds.

- If the simulation time is specified as zero, the simulator performs the scheduled action at the start of simulation.
- If you specify both `startTime` and `periodicity`, the simulator performs the scheduled action at the specified simulation time and repeats the action until the end of simulation. The time interval for repeating the action is specified by the periodic time `periodicity`.

`periodicity` — Periodic time

non-negative scalar

Periodic time, specified as a non-negative scalar. The periodic time is the time interval at which the scheduled action repeats. Units are in seconds.

- For periodic actions, the periodicity value must be greater than zero.

- If the periodic time value is 0, the scheduled action is performed whenever the simulator advances in time in response to an event. The events can occur at regular or irregular time intervals. In this case, the simulator ignores the value of `startTime`.

Output Arguments

`actionID` — Unique identifier for action

integer

Unique identifier for action, returned as an integer. You can also use this value to cancel a scheduled action by using the `cancelAction` function.

Version History

Introduced in R2022b

See Also

Objects

`wirelessNetworkSimulator` | `bluetoothLENode` | `bluetoothNode`

Functions

`addChannelModel` | `cancelAction` | `run` | `addNodes`

Topics

“Create, Configure, and Simulate Bluetooth BR Piconet”
“Bluetooth BR Data and Voice Communication with WLAN Signal Interference”
“Simulate Multiple Bluetooth BR Piconets with ACL Traffic”

wirelessNetworkSimulator.getInstance

Get instance of `wirelessNetworkSimulator` object

Syntax

```
networkSimulator = wirelessNetworkSimulator.getInstance()
```

Description

`networkSimulator = wirelessNetworkSimulator.getInstance()` gets an instance of the `wirelessNetworkSimulator` object initialized by using the `wirelessNetworkSimulator.init()` function.

Examples

Get `wirelessNetworkSimulator` Object

Initialize the wireless network simulator. The `wirelessNetworkSimulator.init()` function creates a `wirelessNetworkSimulator` object in the MATLAB workspace.

```
wirelessNetworkSimulator.init();
```

Get an instance of the initialized `wirelessNetworkSimulator` object.

```
networkSimulator = wirelessNetworkSimulator.getInstance();
```

Output Arguments

`networkSimulator` — Wireless network simulator

`wirelessNetworkSimulator` object

Wireless network simulator, returned as a `wirelessNetworkSimulator` object.

Version History

Introduced in R2022b

See Also

Objects

`wirelessNetworkSimulator` | `bluetoothLENode` | `bluetoothNode`

Functions

`addChannelModel` | `scheduleAction` | `cancelAction` | `addNodes` | `run`

Topics

“Create, Configure, and Simulate Bluetooth BR Piconet”

“Bluetooth BR Data and Voice Communication with WLAN Signal Interference”
“Simulate Multiple Bluetooth BR Piconets with ACL Traffic”