

---

# **Blatann Documentation**

***Release v0.5.0***

**Thomas Gerstenberg**

**Mar 02, 2023**



## CONTENTS:

<b>1</b>	<b>Getting Started</b>	<b>3</b>
1.1	Introduction . . . . .	3
1.2	Install . . . . .	3
1.3	Running with macOS brew python . . . . .	3
1.4	Setting up Hardware . . . . .	4
1.5	Smoke Test the Setup . . . . .	4
<b>2</b>	<b>Core Classes</b>	<b>5</b>
2.1	Events . . . . .	5
2.2	Waitables . . . . .	5
2.3	BLE Device . . . . .	6
2.4	Advertising . . . . .	6
2.5	Scanning . . . . .	6
2.6	Peer . . . . .	7
2.7	Security . . . . .	7
2.8	Local GATT Database . . . . .	7
2.9	Remote GATT Database . . . . .	7
<b>3</b>	<b>Examples</b>	<b>9</b>
<b>4</b>	<b>Compatibility Matrix</b>	<b>11</b>
<b>5</b>	<b>Troubleshooting</b>	<b>13</b>
<b>6</b>	<b>Library Architecture</b>	<b>15</b>
6.1	Class Hierarchy (WIP) . . . . .	15
6.2	Threading Model . . . . .	16
<b>7</b>	<b>Changelog</b>	<b>17</b>
7.1	v0.5.0 . . . . .	17
7.2	v0.4.0 . . . . .	18
7.3	v0.3.6 . . . . .	18
7.4	v0.3.5 . . . . .	18
7.5	v0.3.4 . . . . .	19
7.6	v0.3.3 . . . . .	20
7.7	v0.3.2 . . . . .	20
7.8	v0.3.1 . . . . .	21
7.9	v0.3.0 . . . . .	21
<b>8</b>	<b>API Reference</b>	<b>23</b>
8.1	blatann . . . . .	23

<b>9 Indices and tables</b>	<b>167</b>
<b>Python Module Index</b>	<b>169</b>
<b>Index</b>	<b>171</b>

blåtann: Norwegian word for “blue tooth”

Blatann aims to provide a high-level, object-oriented interface for interacting with bluetooth devices through python. It operates using a Nordic nRF52 through Nordic’s `pc-ble-driver-py` library and the associated Connectivity firmware for the device.



## GETTING STARTED

As of v0.3.0, blatann will only support Python 3.7+. v0.2.x will be partially maintained for Python 2.7 by backporting issues/bugs found in 0.3.x.

### 1.1 Introduction

This library relies on a Nordic nRF52 connected via USB to the PC and flashed with the Nordic Connectivity firmware in order to operate.

---

**Note:** This library will not work as a driver for any generic Bluetooth HCI USB device nor built-in Bluetooth radios. The driver is very specific to Nordic and their provided connectivity firmware, thus other Bluetooth vendors will not work. (BLE communications with non-Nordic devices is not affected.)

---

Below are the known supported devices:

- nRF52832 Development Kit (PCA10040)
- nRF52840 Development Kit (PCA10056)
- nRF52840 USB Dongle (PCA10059)

### 1.2 Install

Blatann can be installed through pip: `pip install blatann`

### 1.3 Running with macOS brew python

`pc-ble-driver-py` consists of a shared object which is linked to mac's system python. In order to use it with brew's python install, you'll need to run `install_name_tool` to modify the `.so` to point to brew python instead of system python.

Example shell script to do so (with more info) can be found here: [macos script](#)

## 1.4 Setting up Hardware

Once one of the hardware devices above is connected via USB, the Nordic Connectivity firmware can be flashed using Nordic's [nRF Connect](#) Application. There are other methods you can use (such as `nrftool`), however this is the least-complicated way. Further instructions for using nRF Connect are out of scope for this as Nordic has great documentation for using their app already.

The firmware image to use can be found within the installed `pc-ble-driver-py` python package under the `hex/` directory. From there, it's a drag and drop operation to get the firmware image onto the hardware.

See the [Compatibility Matrix](#) which lists what software, firmware, and hardware components work together.

## 1.5 Smoke Test the Setup

Once the hardware is flashed and Blatann is installed, the Scanner example can be executed to ensure everything is working. Blatann's examples can be executed from the command line using

```
python -m blatann.examples <example_name> <comport>
```

For the smoke test, use the `scanner` example which will stream any advertising packets found for about 4 seconds:

```
python -m blatann.examples scanner <comport>
```

If everything goes well, head on over to [Examples](#) to look at the library in action or visit [Library Architecture](#) to get an overview of the library. If things do not seem to be working, check out the [Troubleshooting](#) page.



## CORE CLASSES

Below are quick descriptions and links to the primary/core classes that are used to perform the various Bluetooth operations. This information, including more thorough usage, can be found within example code ([Examples](#))

### 2.1 Events

The [Event](#) type is the basic building block of the blatann library.

Bluetooth operations are inherently asynchronous, thus asynchronous events must be used in order to communicate when things happen.

Almost all of the classes below implement one or more Events which can have multiple handler functions registered to process incoming data. Event properties are commonly named in the format of `on_*`, such as `on_timeout` or `on_read_complete`. The event properties also document the parameter types that the handler should accept. Majority of the events emit two parameters, a sender parameter, which provides the event source, and an `event_args` parameter, which provides the data associated with the event. Those familiar with C#/.NET, this should look very similar.

```
def my_handler(sender, event_args):  
    # Handle the event  
some_object.on_some_event.register(my_handler)
```

### 2.2 Waitables

[Waitable](#), [EventWaitable](#)

Waitables are the solution to providing an API which supports synchronous, procedural code given the asynchronous nature of Bluetooth. For every asynchronous Bluetooth operation that is performed a `Waitable` object is returned which the user can then `wait()` on to block the current thread until the operation completes.

```
sender, event_args = characteristic.read().wait(timeout=5)
```

---

**Note:** Take care to not call `wait()` within an event handler as the system will deadlock (see Threading section under [Library Architecture](#) for more info).

---

Asynchronous paradigms are also supported through waitables where the user can register a handler to be called when the operation completes:

```
def my_characteristic_read_handler(sender, event_args):  
    # Handle read complete  
    characteristic.read().then(my_characteristic_read_handler)
```

## 2.3 BLE Device

The *BleDevice* represents Nordic Bluetooth microcontroller itself. It is the root object of everything within this library.

To get started, instantiate a *BleDevice* and open it:

```
from blatann import BleDevice  
  
ble_device = BleDevice("COM1")  
ble_device.configure()  
ble_device.open()  
# Ready to use
```

The BLE Device is also responsible for initiating connections to peripheral devices and managing the local GATT database.

## 2.4 Advertising

The *Advertiser* component is accessed through the `ble_device.advertiser` attribute. It is configured using *AdvertisingData* objects to set the payloads to advertise

```
from blatann.gap.advertising import AdvertisingData  
adv_data = AdvertisingData(flags=0x06, local_name="My Name")  
scan_data = AdvertisingData(service_uuid16s="123F")  
ble_device.advertiser.set_advertise_data(adv_data, scan_data)  
ble_device.advertiser.start(adv_interval_ms=50)
```

## 2.5 Scanning

The *Scanner* component is accessed through the `ble_device.scanner` attribute.

The scanner output consists of a *ScanReportCollection*, which is comprised of *ScanReport* objects that represent advertising packets discovered.

```
scan_report_collection = ble_device.scanner.start_scan().wait(timeout=20)
```

## 2.6 Peer

The *Peer* class represents a Bluetooth connection to another device.

For connections as a peripheral to a central device, this peer object is static and accessed via the `ble_device.client` attribute. For connections as a central to a peripheral device, the peer is created as a result of *BleDevice.connect*.

Regardless of the connection type, the Peer is the basis for any connection-oriented Bluetooth operation, such as configuring the MTU, discovering databases, reading/writing characteristics, etc.

```
# Connect to a peripheral and exchange MTU
peer = ble_device.connect(peer_address).wait()
peer.exchange_mtu(144).wait()
# Exchange the MTU with a client
ble_device.client.exchange_mtu(183).wait()
```

## 2.7 Security

The processes for pairing and bonding is managed by a peer's *SecurityManager*, accessed via the `peer.security` attribute.

## 2.8 Local GATT Database

The *GattsDatabase* is accessed through the `ble_device.database` attribute. The database holds all of the services and characteristics that can be discovered and interacted with by a client.

*GattsService* s can be added to the database and *GattsCharacteristic* s are added to the services. The primary interaction point is through characteristics, which provides methods for setting values, handling writes, and notifying values to the client.

## 2.9 Remote GATT Database

The peer's *GattcDatabase* is accessed through the `peer.database` attribute. The database is populated through the *peer.discover\_services* procedure. From there, the Peer's *GattcCharacteristic* s can be read, written, and subscribed to.



## EXAMPLES

*This section is a work in progress. Still need to add specific sections giving an overview for each example*

Examples can be found here: [Blatann examples](#)



## COMPATIBILITY MATRIX

Table 1: Software/Firmware Compatibility Matrix

Blatann Version	Python Version	Connectivity Firmware Version	SoftDevice Version	pc-ble-driver-py Version
v0.2.x	2.7 Only	v1.2.x	v3	<= 0.11.4
v0.3+	3.7+	v4.1.x	v5	>= 0.12.0

Firmware images are shipped within the `pc-ble-driver-py` package under the `hex/` directory. Below maps which firmware images to use for which devices. For Blatann v0.2.x, firmware images are under subdir `sd_api_v3`. For Blatann v0.3+, firmware images are under subdir `sd_api_v5`.

Table 2: Firmware/Hardware Compatibility Matrix

Hardware	Firmware Image
nRF52832 Devkit	<code>connectivity_x.y.z_&lt;baud&gt;_with_s132_x.y.hex</code> (note the baud rate in use!)
nRF52840 Devkit	<code>connectivity_x.y.z_&lt;baud&gt;_with_s132_x.y.hex</code> (note the baud rate in use!) <b>or</b> <code>connectivity_x.y.z_usb_with_s132_x.y.hex</code> if using the USB port on the side
nRF52840 USB Dongle	<code>connectivity_x.y.z_usb_with_s132_x.y.hex</code>

---

**Note:** Blatann provides a default setting for the baud rate to use with the device. For 0.2.x, the default baud is 115200 whereas 0.3+ the default is 1M (and USB doesn't care). This is only an issue when running examples through the command line as it doesn't expose a setting for the baud rate. When writing your own script, it can be configured however it's needed.

---





## **TROUBLESHOOTING**

*This section is a work in progress*

### **General Debugging**

Blatann uses the built-in logging module to log all events and driver calls. The library also contains a helper function to configure/enable: `blatann.utils.setup_logger()`.

When submitting an issue, please include logs of the behavior at the DEBUG level.

### **Specific Error Messages**

Error message Failed to open. Error code: 0x8029 - Check your comport settings (baud, port, etc.).

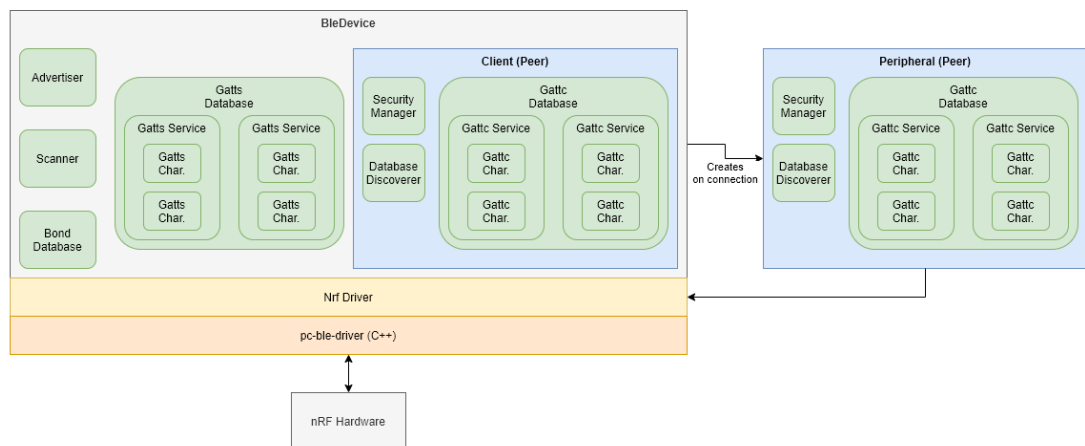
Note that the nRF52840 USB dongle will enumerate 2 separate ports: one for the bootloader during flashing and one for the application. Make sure that you check the port number after the bootloader exits and application starts.



## LIBRARY ARCHITECTURE

### 6.1 Class Hierarchy (WIP)

Very high-level diagram outlining major/public components



#### BleDevice

The *BleDevice* class represents the Nordic hardware itself and is the primary entry point for the library. It provides the high-level APIs for all BLE operations, such as advertising, scanning, and connections. It also manages the device's configuration and bonding database.

#### Advertiser

The *Advertiser* class provides the API for setting advertising data, intervals, and starting/stopping of advertising. It is accessed via `BleDevice::attr:~blatann.device.BleDevice.advertiser` attribute.

#### Scanner

The *Scanner* class provides the API for scanning for advertising packets. Scan reports are emitted during scanning and can be used to initiate connections with the advertising peripherals. It is accessed via `BleDevice::attr:~blatann.device.BleDevice.scanner` attribute.

#### Peer

The *Peer* class represents a connection with another device over Bluetooth. The BLE Device contains a single *Client* object, which is the connection with a Client/Central device. When connecting to Server/Peripheral devices as a

Client/Central via `BleDevice.connect()`, a *Peripheral* object is created and returned as a result of the `connect()` call.

The Peer object provides everything necessary for communications with the device, including security, database creation (as a server) and discovery (as a client), connection parameters, etc.

## 6.2 Threading Model

Most BLE operations are inherently asynchronous. The Nordic has a combination of synchronous function calls and asynchronous events. Synchronous function calls may not return the result immediately and instead return within an asynchronous event as to not block the main context.

Blatann uses a second python thread for handling asynchronous events received over BLE. This event thread (named "<comport>Thread") handles a queue of events received from the C++ Driver and dispatches them to the registered callbacks. These callbacks are the triggers for the various *Event* objects that exist throughout the Peer, Scanner, Advertiser, and other objects.

In order to support both both event-driven and procedural styles of programming, a mechanism needs to exist in order to communicate events back to the main thread so asynchronous functionality (such as characteristic reads/writes) can be made synchronous. The result of this is the *Waitable* class.

Asynchronous method calls in the library will return a *Waitable* object which can either then have callbacks registered (to keep things asynchronous) or waited on (with or without timeout) from the main thread to make it synchronous. This is a very similar concept to `concurrent.futures.Future`, just a different implementation.

Since there is only a single thread which handles all events, **do not call `Waitable.wait()` within an event handler as it will cause a deadlock**. Calling BLE methods from the event handler context is perfectly fine and can use `Waitable.then(callback)` to handle the result of the operation asynchronously.

## CHANGELOG

### 7.1 v0.5.0

v0.5.0 reworks bonding database to JSON, adds a few features, and fixes a few bugs. Full list of issues and PRs for this release can be found here: [0.5.0 Milestone](#)

#### Highlights

- Adds support for the scanner to resolve peer addresses.
  - *ScanReport* has 2 new properties: `is_bonded_device: bool` and `resolved_address: Optional[PeerAddress]`
- Adds support for privacy settings to advertise with a private resolvable or non-resolvable address
- Adds parameter to device configuration to set a different connection event length

#### Fixes

- Fixes incorrect variable name when a queued GATT operation times out (thanks @klow68)
- Fixes incorrect key length when converting an LESC private key to a raw bytearray (thanks @klow68). Function is unused within blatann
- Fixes issue where the service changed characteristic was not correctly getting added to the GATT server when configured

#### Changes

- Reworks the bond database to be saved using JSON instead of pickle.
  - Existing "system" and "user" database files configured in the BLE device will automatically be migrated to JSON
  - Other database files configured by filepath will continue to use pickle and can be updated manually using *migrate\_bond\_database()*
- Bond DB entries will now save the local BLE address that was used to generate the bonding data.
  - This will allow multiple nRF boards to use the same DB file and not resolve bond entries if it was not created with that board/address. This fixes potential issues where restoring a connection to a peer that was bonded to a different nRF board can cause the local device to think it has a bond, however the peer has bond info with a different, mismatched address.
- Moves bond-related resolve logic out of the security manager and into the bond database

## 7.2 v0.4.0

v0.4.0 introduces some new features and fixes a couple of issues. Full list of issues and PRs for this release can be found here: [0.4.0 Milestone](#)

### Highlights

- Adds support for reading RSSI of an active connection, plus example usage of API
- Adds Event+Waitable for Connection Parameter Update procedures
  - Additionally adds support for accepting/rejecting update requests as a central
- Adds support for setting the device's transmit power
- Adds support for setting advertising channel masks

### Fixes

- Fixes issues seen when performing certain pairing routines on linux
- Fixes for misc. advertising corner cases
- Fixes an issue with BasicGlucoseDatabase introduced in the python 3 migration

## 7.3 v0.3.6

v0.3.6 is a minor bugfix update and some small improvements

### Fixes

- Fixes an uncaught exception caused when handling a failed bond database load (thanks @dkkeller)
- Fixes an issue where waiting on indications to be confirmed did not work. Regression introduced in v0.3.4

### Changes

- Updates the descriptor discovery portion of service discovery to be more efficient, speeding up service discovery times
- Updates the API lock at the driver layer to be per-device. This will reduce lock contention when using multiple BLE Devices in different threads

## 7.4 v0.3.5

v0.3.5 is a small update that primarily provides some bug fixes and cleanup to the bonding process.

### Highlights

- Overall increased stability when restoring encryption using long-term keys for a previously-bonded device
- Adds param to set the CCCD write security level for a characteristic

### Fixes

- Restoring legacy bonding LTKs as a central now works correctly

### Changes

- [Issue 60](#) - The default bonding database file has been moved into the user directory instead of within the package contents (`~/.blatann/bonding_db.pkl`).

- An optional parameter has been added to the `BleDevice` constructor for specifying the file to use for convenience
- To revert to the previous implementation, specify `bond_db_filename="system"` when creating the `BleDevice` object
- To use the new storage location but keep the bonding data from previous version, copy over the database file from `<blatann_install_loc>/.user/bonding_db.pkl` to the location noted above

## 7.5 v0.3.4

v0.3.4 brings several new features (including characteristic descriptors) and a couple bug fixes. A fairly large refactoring of the GATT layer took place to make room for the descriptors, however no public-facing APIs were modified.

### Highlights

- [Issue 11](#) - Adds support for adding descriptor attributes to characteristics
  - See the [Central Descriptor Example](#) and [Peripheral Descriptor Example](#) for how they can be used
- Adds a new `bt_sig` sub-package which provides constants and UUIDs defined by Bluetooth SIG.
- Adds visibility to the device's Generic Access Service: `BleDevice.generic_access_service`
  - Example usage has been added to the peripheral example
- Adds support for performing PHY channel updates
  - **Note:** Coded PHY is currently not supported, only 1Mbps and 2Mbps PHYs
- Adds a description attribute to the UUID class. The standard UUIDs have descriptions filled out, custom UUIDs can be set by the user.

### Fixes

- Fixes an issue with bonding failing on linux
- Fixes an issue where the `sys_attr_missing` event was not being handled
- Adds missing low-level error codes for the RPC layer
- Fixes race condition when waiting on ID-based events causing an `AttributeError`. Event subscription previously occurred before the ID was set and there was a window where the callback could be triggered before the ID was set in the object instance. This issue was most prominent after introducing the write/notification queuing changes in combination with a short connection interval.

### Changes

- The `device_name` parameter has been removed from `BleDevice.configure()`. This wasn't working before and has been added into the Generic Access Service.
- Write, notification, and indication queuing has been tweaked such that non-ack operations (write w/o response, notifications) now take advantage of a hardware queue independent of the acked counterparts (write request, indications)
- Service discovery was modified to allow descriptor discovery and in some cases (depending on peripheral stack) run faster
- `DecodedReadWriteEventDispatcher` has been moved from `blatann.services` to `blatann.services.decoded_event_dispatcher`. This was to solve a circular dependency issue once new features were added in.

- The glucose service has been updated to make better use of the notification queuing mechanism. Glucose record transmission is sped up greatly

## 7.6 v0.3.3

v0.3.3 fixes a couple issues and adds some enhancements to the security manager.

### Highlights

- Adds handling for peripheral-initiated security/pairings
- Adds finer control over accepting/rejecting pairing requests based on the peer's role, whether or not it's already bonded, etc.
- Adds more events and properties to expose the connection's security state
- Adds method to delete a connected peer's bonding data for future connections

### Fixes

- Fixes issue where the length of the scan response payload was not correctly being checked against the maximum 31-byte length
- Fixes issue that was not allowing central devices to initiate encryption to an already-bonded peripheral device
- Fixes issue that wasn't allowing time to be read from the Current Time service as a client

### Changes

- Advertising payloads received that are padded with 0's at the end are now ignored and do not produce spammy logs
- Adds a device-level method to set the default security level to use for all subsequent connections to peripheral devices
- Adds a name property to the `Peer` class. This is auto-populated from the scan report (if connecting to a peripheral) and can be set manually if desired.

## 7.7 v0.3.2

v0.3.2 is a bug fix release

### Fixes

- [Issue 40](#) - Fixes issue where service discovery fails if the server returns `attribute_not_found` while discovering services
- [Issue 42](#) - Fixes issue where `Advertiser.is_advertising` could return false if `auto_restart` is enabled and advertising times out

### Added Features

- Exposes a new `Advertiser.auto_restart` property so it can be get/set outside of `Advertiser.start()`



## 7.8 v0.3.1

v0.3.1 provides a few enhancements and features from the previous release.

### Highlights

- Adds the ability to discover, read, and write a connected central device's GATT database as a peripheral.
  - Example usage has been added to the peripheral example where it will discover the connected device's database after pairing completes
  - **NOTE:** The inverse of this should be considered experimental (i.e. acting as a central and having a peripheral read/write the local database).
- Adds the ability to perform writes without responses, both as a client and as a peripheral
  - New APIs have been added to the `GattcCharacteristic` class: `write_without_response()` and `writable_without_response`
- Adds API to trigger data length update procedures (with corresponding event) on the `Peer` class
  - The API does not allow the user to select a data length to use, i.e. the optimal data length is chosen by the SoftDevice firmware

### Changes

- The connection event length has been updated to support the max-length DLE value (251bytes) at the shortest connection interval (7.5ms)
- Updates to documentation and type hinting
- Minor changes to logging, including removing spammy/duplicate logs when numerous characteristics exist in the GATT database

### Fixes

- Fixes issue where iterating over the scan report in real-time was not returning the recently read packet and instead was returning the combined packet for the device's address. This was causing duplicate packets to not be marked in the scanner example.

## 7.9 v0.3.0

v0.3.0 marks the first stable release for Python 3.7+.

Unfortunately a comprehensive changelog is not available for this release as a lot went in to migrate to Py3/Softdevice v5. That said, public API should be mostly unchanged except for the noted changes below.

### Highlights

- Python 3.7+ only
- Requires `pc-ble-driver-py` v0.12.0+
- Requires Nordic Connectivity firmware v4.1.1 (Softdevice v5)

### Changes

- `Scanner.scanning` field was replaced with read-only property `Scanner.is_scanning`
- Parameter validation was added for Advertising interval, Scan window/interval/timeout, and connection interval/timeout.
  - Will raise `ValueError` exceptions when provided parameters are out of range

- With Python 3, converting from `bytes` to `str` (and vice-versa) requires an encoding format. By default, the encoding scheme is `utf-8` and can be set per-characteristic using the `string_encoding` property
- `peer.disconnect()` will now always return a `Waitable` object. Before it would return `None` if not connected to the peer. If `disconnect()` is called when the peer is not connected, it will return a `Waitable` object that expires immediately

### Fixes

- Fixes an issue where unsubscribing from a driver event while processing the event was causing the the next handler for the driver event to be skipped
  - Back-ported to v0.2.9

### Features

(This list is not comprehensive)

- Driver now property works with 2 devices simultaneously
- Event callbacks can now be used in a `with` context so the handler can be deregistered at the end of a block
  - [Event callback example](#)
- The `ScanFinishedWaitable` now provides a `scan_reports` iterable which can be used to iterate on advertising packets as they're seen in real-time
  - [ScanFinishedWaitable example](#)
- The `Peer` object now exposes properties for the active connection parameters and configured/preferred connection parameters
- The `Peripheral` object exposes an `on_service_discovery_complete` event
- Added `AdvertisingData.to_bytes()` to retrieve the data packet that will be advertised over the air

## API REFERENCE

### 8.1 blatann

#### 8.1.1 blatann package

##### Subpackages

##### blatann.bt\_sig package

##### Submodules

##### blatann.bt\_sig.assigned\_numbers module

```
class blatann.bt_sig.assigned_numbers.Format(, description="")
    Bases: IntEnumWithDescription
    Format enumeration for use with the blatann.gatt.PresentationFormat class
    rfu = 0
    boolean = 1
    twobit = 2
    nibble = 3
    uint8 = 4
    uint12 = 5
    uint16 = 6
    uint24 = 7
    uint32 = 8
    uint48 = 9
    uint64 = 10
    uint128 = 11
    sint8 = 12
```

```
sint12 = 13
sint16 = 14
sint24 = 15
sint32 = 16
sint48 = 17
sint64 = 18
sint128 = 19
float32 = 20
float64 = 21
sfloat = 22
float = 23
duint16 = 24
utf8s = 25
utf16s = 26
struct = 27
```

```
class blatann.bt_sig.assigned_numbers.Namespace(, description=")
```

Bases: [\*IntEnumWithDescription\*](#)

Namespace enumeration for use with the [\*blatann.gatt.PresentationFormat\*](#) class

```
unknown = 0
```

```
bt_sig = 1
```

```
class blatann.bt_sig.assigned_numbers.NamespaceDescriptor(, description=")
```

Bases: [\*IntEnumWithDescription\*](#)

Namespace descriptor enumeration for use with the [\*blatann.gatt.PresentationFormat\*](#) class

```
auxiliary = 264
```

```
back = 257
```

```
backup = 263
```

```
bottom = 259
```

```
external = 272
```

```
flash = 266
```

```
front = 256
```

```
inside = 267
```

```
internal = 271
```

```

left = 269
lower = 261
main = 262
outside = 268
right = 270
supplementary = 265
top = 258
unknown = 0
upper = 260

```

```

class blatann.bt_sig.assigned_numbers.Units(_, description="")
    Bases: IntEnumWithDescription
    Units enumeration for use with the blatann.gatt.PresentationFormat class
    unitless = 9984

    absorbed_dose_gray = 10035
    absorbed_dose_rate_gray_per_second = 10068
    acceleration_metres_per_second_squared = 10003
    activity_referred_to_a_radionuclide_becquerel = 10034
    amount_concentration_mole_per_cubic_metre = 10010
    amount_of_substance_mole = 9990
    angular_acceleration_radian_per_second_squared = 10052
    angular_velocity_radian_per_second = 10051
    angular_velocity_revolution_per_minute = 10152
    area_barn = 10116
    area_hectare = 10086
    area_square_metres = 10000
    capacitance_farad = 10025
    catalytic_activity_concentration_katal_per_cubic_metre = 10071
    catalytic_activity_katal = 10037
    concentration_count_per_cubic_metre = 10165
    concentration_parts_per_billion = 10181
    concentration_parts_per_million = 10180

```

```
current_density_ampere_per_square_metre = 10008
density_kilogram_per_cubic_metre = 10005
dose_equivalent_sievert = 10036
dynamic_viscosity_pascal_second = 10048
electric_charge_ampere_hours = 10160
electric_charge_coulomb = 10023
electric_charge_density_coulomb_per_cubic_metre = 10060
electric_conductance_siemens = 10027
electric_current_ampere = 9988
electric_field_strength_volt_per_metre = 10059
electric_flux_density_coulomb_per_square_metre = 10062
electric_potential_difference_volt = 10024
electric_resistance_ohm = 10026
energy_density_joule_per_cubic_metre = 10058
energy_gram_calorie = 10153
energy_joule = 10021
energy_kilogram_calorie = 10154
energy_kilowatt_hour = 10155
exposure_coulomb_per_kilogram = 10067
force_newton = 10019
frequency_hertz = 10018
heat_capacity_joule_per_kelvin = 10054
heat_flux_density_watt_per_square_metre = 10053
illuminance_lux = 10033
inductance_henry = 10030
irradiance_watt_per_square_metre = 10166
length_foot = 10147
length_inch = 10146
length_metre = 9985
length_mile = 10148
length_nautical_mile = 10115
```

```
length_parsec = 10145
length_yard = 10144
length_angstrom = 10114
logarithmic_radio_quantity_bel = 10119
logarithmic_radio_quantity_neper = 10118
luminance_candela_per_square_metre = 10012
luminous_efficacy_lumen_per_watt = 10174
luminous_energy_lumen_hour = 10175
luminous_exposure_lux_hour = 10176
luminous_flux_lumen = 10032
luminous_intensity_candela = 9991
magnetic_field_strength_ampere_per_metre = 10009
magnetic_flux_density_tesla = 10029
magnetic_flux_weber = 10028
mass_concentration_kilogram_per_cubic_metre = 10011
mass_density_milligram_per_decilitre = 10161
mass_density_millimole_per_litre = 10162
mass_flow_gram_per_second = 10177
mass_kilogram = 9986
mass_pound = 10168
mass_tonne = 10088
metabolic_equivalent = 10169
molar_energy_joule_per_mole = 10065
molar_entropy_joule_per_mole_kelvin = 10066
moment_of_force_newton_metre = 10049
per_mille = 10158
percentage = 10157
period_beats_per_minute = 10159
permeability_henry_per_metre = 10064
permittivity_farad_per_metre = 10063
plane_angle_degree = 10083
```

```
plane_angle_minute = 10084
plane_angle_radian = 10016
plane_angle_second = 10085
power_watt = 10022
pressure_bar = 10112
pressure_millimetre_of_mercury = 10113
pressure_pascal = 10020
pressure_pound_force_per_square_inch = 10149
radiance_watt_per_square_metre_steradian = 10070
radiant_intensity_watt_per_steradian = 10069
refractive_index = 10013
relative_permeability = 10014
solid_angle_steradian = 10017
sound_pressure_decibel_spl = 10179
specific_energy_joule_per_kilogram = 10056
specific_heat_capacity_joule_per_kilogram_kelvin = 10055
specific_volume_cubic_metre_per_kilogram = 10007
step_per_minute = 10170
stroke_per_minute = 10172
surface_charge_density_coulomb_per_square_metre = 10061
surface_density_kilogram_per_square_metre = 10006
surface_tension_newton_per_metre = 10050
thermal_conductivity_watt_per_metre_kelvin = 10057
thermodynamic_temperature_degree_celsius = 10031
thermodynamic_temperature_degree_fahrenheit = 10156
thermodynamic_temperature_kelvin = 9989
time_day = 10082
time_hour = 10081
time_minute = 10080
time_month = 10164
time_second = 9987
```



```
time_year = 10163
transfer_rate_milliliter_per_kilogram_per_minute = 10167
velocity_kilometer_per_minute = 10173
velocity_kilometre_per_hour = 10150
velocity_knot = 10117
velocity_metres_per_second = 10002
velocity_mile_per_hour = 10151
volume_cubic_metres = 10001
volume_flow_litre_per_second = 10178
volume_litre = 10087
wavenumber_reciprocal_metre = 10004

class blatann.bt_sig.assigned_numbers.Appearance(
    _, description=""
)
    Bases: IntEnumWithDescription
    Appearance enumeration for use with advertising data
    unknown = 0
    phone = 64
    computer = 128
    watch = 192
    sports_watch = 193
    clock = 256
    display = 320
    remote_control = 384
    eye_glasses = 448
    tag = 512
    keyring = 576
    media_player = 640
    barcode_scanner = 704
    thermometer = 768
    thermometer_ear = 769
    heart_rate_sensor = 832
    heart_rate_sensor_heart_rate_belt = 833
```

```
blood_pressure = 896
blood_pressure_arm = 897
blood_pressure_wrist = 898
hid = 960
hid_keyboard = 961
hid_mouse = 962
hid_joystick = 963
hid_gamepad = 964
hid_digitizer = 965
hid_card_reader = 966
hid_digital_pen = 967
hid_barcode = 968
glucose_meter = 1024
running_walking_sensor = 1088
running_walking_sensor_in_shoe = 1089
running_walking_sensor_on_shoe = 1090
running_walking_sensor_on_hip = 1091
cycling = 1152
cycling_cycling_computer = 1153
cycling_speed_sensor = 1154
cycling_cadence_sensor = 1155
cycling_power_sensor = 1156
cycling_speed_cadence_sensor = 1157
pulse_oximeter = 3136
pulse_oximeter_fingertip = 3137
pulse_oximeter_wrist_worn = 3138
weight_scale = 3200
outdoor_sports_act = 5184
outdoor_sports_act_loc_disp = 5185
outdoor_sports_act_loc_and_nav_disp = 5186
outdoor_sports_act_loc_pod = 5187
```

```
outdoor_sports_act_loc_and_nav_pod = 5188
as_bytes()
```

### blatann.bt\_sig.uuids module

Bluetooth SIG defined UUIDs, populated from their website.

See also:

<https://btprodspecificationrefs.blob.core.windows.net/assigned-values/16-bit%20UUID%20Numbers%20Document.pdf>

Definitions last scraped on 2022/05/02

**class** blatann.bt\_sig.uuids.DeclarationUuid

Bases: `object`

UUIDs used for declarations within the GATT Database

**primary\_service** = 2800

**secondary\_service** = 2801

**include** = 2802

**characteristic** = 2803

**class** blatann.bt\_sig.uuids.DescriptorUuid

Bases: `object`

UUIDs that are used for characteristic descriptors

**extended\_properties** = 2900

**user\_description** = 2901

**cccd** = 2902

**sccd** = 2903

**presentation\_format** = 2904

**aggregate\_format** = 2905

**valid\_range** = 2906

**external\_report\_reference** = 2907

**report\_reference** = 2908

**number\_of\_digitals** = 2909

**value\_trigger\_setting** = 290a

**es\_configuration** = 290b

**es\_measurement** = 290c

**es\_trigger\_setting** = 290d

```
time_trigger_setting = 290e
complete_br_edr_transport_block_data = 290f
class blatann.bt_sig.uuids.ServiceUuid
    Bases: object
    Bluetooth SIG defined service UUIDs
    alert_notification = 1811
    audio_input_control = 1843
    audio_stream_control = 184e
    automation_io = 1815
    basic_audio_announcement = 1851
    battery_service = 180f
    binary_sensor = 183b
    blood_pressure = 1810
    body_composition = 181b
    bond_management = 181e
    broadcast_audio_announcement = 1852
    broadcast_audio_scan = 184f
    common_audio = 1853
    constant_tone_extension = 184a
    continuous_glucose_monitoring = 181f
    coordinated_set_identification = 1846
    current_time = 1805
    cycling_power = 1818
    cycling_speed_and_cadence = 1816
    device_information = 180a
    device_time = 1847
    emergency_configuration = 183c
    environmental_sensing = 181a
    fitness_machine = 1826
    generic_access = 1800
    generic_attribute = 1801
```

generic\_media\_control = 1849  
generic\_telephone\_bearer = 184c  
glucose = 1808  
health\_thermometer = 1809  
hearing\_access = 1854  
heart\_rate = 180d  
http\_proxy = 1823  
human\_interface\_device = 1812  
immediate\_alert = 1802  
indoor\_positioning = 1821  
insulin\_delivery = 183a  
internet\_protocol\_support = 1820  
link\_loss = 1803  
location\_and\_navigation = 1819  
media\_control = 1848  
mesh\_provisioning = 1827  
mesh\_proxy = 1828  
microphone\_control = 184d  
next\_dst\_change = 1807  
object\_transfer = 1825  
phone\_alert\_status = 180e  
physical\_activity\_monitor = 183e  
published\_audio\_capabilities = 1850  
pulse\_oximeter = 1822  
reconnection\_configuration = 1829  
reference\_time\_update = 1806  
running\_speed\_and\_cadence = 1814  
scan\_parameters = 1813  
telephone\_bearer = 184b  
tmas = 1855  
transport\_discovery = 1824

```
tx_power = 1804
```

```
user_data = 181c
```

```
volume_control = 1844
```

```
volume_offset_control = 1845
```

```
weight_scale = 181d
```

```
class blatann.bt_sig.uuids.CharacteristicUuid
```

```
    Bases: object
```

```
    Bluetooth SIG defined characteristic UUIDs
```

```
    active_preset_index = 2bdc
```

```
    activity_current_session = 2b44
```

```
    activity_goal = 2b4e
```

```
    adv_constant_tone_interval = 2bb1
```

```
    adv_constant_tone_min_length = 2bae
```

```
    adv_constant_tone_min_tx_count = 2baf
```

```
    adv_constant_tone_phy = 2bb2
```

```
    adv_constant_tone_tx_duration = 2bb0
```

```
    aerobic_heart_rate_lower_limit = 2a7e
```

```
    aerobic_heart_rate_upper_limit = 2a84
```

```
    aerobic_threshold = 2a7f
```

```
    age = 2a80
```

```
    aggregate = 2a5a
```

```
    alert_category_id = 2a43
```

```
    alert_category_id_bit_mask = 2a42
```

```
    alert_level = 2a06
```

```
    alert_notification_control_point = 2a44
```

```
    alert_status = 2a3f
```

```
    altitude = 2ab3
```

```
    ammonia_concentration = 2bcf
```

```
    anaerobic_heart_rate_lower_limit = 2a81
```

```
    anaerobic_heart_rate_upper_limit = 2a82
```

```
    anaerobic_threshold = 2a83
```

analog = 2a58  
analog\_output = 2a59  
apparent\_wind\_direction = 2a73  
apparent\_wind\_speed = 2a72  
appearance = 2a01  
ase\_control\_point = 2bc6  
audio\_input\_control\_point = 2b7b  
audio\_input\_description = 2b7c  
audio\_input\_state = 2b77  
audio\_input\_status = 2b7a  
audio\_input\_type = 2b79  
audio\_location = 2b81  
audio\_output\_description = 2b83  
available\_audio\_contexts = 2bcd  
average\_current = 2ae0  
average\_voltage = 2ae1  
barometric\_pressure\_trend = 2aa3  
battery\_level = 2a19  
battery\_level\_state = 2a1b  
battery\_power\_state = 2a1a  
bearer\_list\_current\_calls = 2bb9  
bearer\_provider\_name = 2bb3  
bearer\_signal\_strength = 2bb7  
bearer\_signal\_strength\_reporting\_interval = 2bb8  
bearer\_technology = 2bb5  
bearer\_uci = 2bb4  
bearer\_uri\_schemes\_supported\_list = 2bb6  
blood\_pressure\_feature = 2a49  
blood\_pressure\_measurement = 2a35  
blood\_pressure\_record = 2b36  
bluetooth\_sig\_data = 2b39

body\_composition\_feature = 2a9b  
body\_composition\_measurement = 2a9c  
body\_sensor\_location = 2a38  
bond\_management\_control\_point = 2aa4  
bond\_management\_feature = 2aa5  
boolean = 2ae2  
boot\_keyboard\_input\_report = 2a22  
boot\_keyboard\_output\_report = 2a32  
boot\_mouse\_input\_report = 2a33  
br\_edr\_handover\_data = 2b38  
broadcast\_audio\_scan\_control\_point = 2bc7  
broadcast\_receive\_state = 2bc8  
bss\_control\_point = 2b2b  
bss\_response = 2b2c  
call\_control\_point = 2bbe  
call\_control\_point\_optional\_opcodes = 2bbf  
call\_friendly\_name = 2bc2  
call\_state = 2bbd  
caloric\_intake = 2b50  
carbon\_monoxide\_concentration = 2bd0  
cardiorespiratory\_activity\_instantaneous\_data = 2b3e  
cardiorespiratory\_activity\_summary\_data = 2b3f  
central\_address\_resolution = 2aa6  
cgm\_feature = 2aa8  
cgm\_measurement = 2aa7  
cgm\_session\_run\_time = 2aab  
cgm\_session\_start\_time = 2aaa  
cgm\_specific\_ops\_control\_point = 2aac  
cgm\_status = 2aa9  
chromatic\_distance\_from\_planckian = 2ae3  
chromaticity\_coordinate = 2b1c



chromaticity\_coordinates = 2ae4  
chromaticity\_in\_cct\_and\_duv\_values = 2ae5  
chromaticity\_tolerance = 2ae6  
cie\_color\_rendering\_index = 2ae7  
client\_supported\_features = 2b29  
coefficient = 2ae8  
constant\_tone\_extension\_enable = 2bad  
content\_control\_id = 2bba  
coordinated\_set\_size = 2b85  
correlated\_color\_temperature = 2ae9  
count\_16 = 2aea  
count\_24 = 2aeb  
country\_code = 2aec  
cross\_trainer\_data = 2ace  
csc\_feature = 2a5c  
csc\_measurement = 2a5b  
current\_group\_object\_id = 2ba0  
current\_time = 2a2b  
current\_track\_object\_id = 2b9d  
current\_track\_segments\_object\_id = 2b9c  
cycling\_power\_control\_point = 2a66  
cycling\_power\_feature = 2a65  
cycling\_power\_measurement = 2a63  
cycling\_power\_vector = 2a64  
database\_change\_increment = 2a99  
database\_hash = 2b2a  
date\_of\_birth = 2a85  
date\_of\_threshold\_assessment = 2a86  
date\_time = 2a08  
date\_utc = 2aed  
day\_date\_time = 2a0a

day\_of\_week = 2a09  
descriptor\_value\_changed = 2a7d  
device\_name = 2a00  
device\_time = 2b90  
device\_time\_control\_point = 2b91  
device\_time\_feature = 2b8e  
device\_time\_parameters = 2b8f  
device\_wearing\_position = 2b4b  
dew\_point = 2a7b  
digital = 2a56  
digital\_output = 2a57  
directory\_listing = 2acb  
dst\_offset = 2a0d  
electric\_current = 2aee  
electric\_current\_range = 2aef  
electric\_current\_specification = 2af0  
electric\_current\_statistics = 2af1  
elevation = 2a6c  
email\_address = 2a87  
emergency\_id = 2b2d  
emergency\_text = 2b2e  
energy = 2af2  
energy\_in\_a\_period\_of\_day = 2af3  
enhanced\_blood\_pressure\_measurement = 2b34  
enhanced\_intermediate\_cuff\_pressure = 2b35  
event\_statistics = 2af4  
exact\_time\_100 = 2a0b  
exact\_time\_256 = 2a0c  
fat\_burn\_heart\_rate\_lower\_limit = 2a88  
fat\_burn\_heart\_rate\_upper\_limit = 2a89  
firmware\_revision\_string = 2a26

first\_name = 2a8a  
fitness\_machine\_control\_point = 2ad9  
fitness\_machine\_feature = 2acc  
fitness\_machine\_status = 2ada  
five\_zone\_heart\_rate\_limits = 2a8b  
fixed\_string\_16 = 2af5  
fixed\_string\_24 = 2af6  
fixed\_string\_36 = 2af7  
fixed\_string\_8 = 2af8  
floor\_number = 2ab2  
four\_zone\_heart\_rate\_limits = 2b4c  
gain\_settings\_attribute = 2b78  
gender = 2a8c  
general\_activity\_instantaneous\_data = 2b3c  
general\_activity\_summary\_data = 2b3d  
generic\_level = 2af9  
global\_trade\_item\_number = 2afa  
glucose\_feature = 2a51  
glucose\_measurement = 2a18  
glucose\_measurement\_context = 2a34  
group\_object\_type = 2bac  
gust\_factor = 2a74  
handedness = 2b4a  
hardware\_revision\_string = 2a27  
hearing\_aid\_features = 2bda  
hearing\_aid\_preset\_control\_point = 2bdb  
heart\_rate\_control\_point = 2a39  
heart\_rate\_max = 2a8d  
heart\_rate\_measurement = 2a37  
heat\_index = 2a7a  
height = 2a8e

```
hid_control_point = 2a4c
hid_information = 2a4a
high_intensity_exercise_threshold = 2b4d
high_resolution_height = 2b47
hip_circumference = 2a8f
http_control_point = 2aba
http_entity_body = 2ab9
http_headers = 2ab7
http_status_code = 2ab8
https_security = 2abb
humidity = 2a6f
idd_annunciation_status = 2b22
idd_command_control_point = 2b25
idd_command_data = 2b26
idd_features = 2b23
idd_history_data = 2b28
idd_record_access_control_point = 2b27
idd_status = 2b21
idd_status_changed = 2b20
idd_status_reader_control_point = 2b24
ieee11073_20601_regulatory_certification_data_list = 2a2a
illuminance = 2afb
incoming_call = 2bc1
incoming_call_target_bearer_uri = 2bbc
indoor_bike_data = 2ad2
indoor_positioning_configuration = 2aad
intermediate_cuff_pressure = 2a36
intermediate_temperature = 2a1e
irradiance = 2a77
language = 2aa2
last_name = 2a90
```

latitude = 2aae  
ln\_control\_point = 2a6b  
ln\_feature = 2a6a  
local\_east\_coordinate = 2ab1  
local\_north\_coordinate = 2ab0  
local\_time\_information = 2a0f  
location\_and\_speed = 2a67  
location\_name = 2ab5  
longitude = 2aaf  
luminous\_efficacy = 2afc  
luminous\_energy = 2afd  
luminous\_exposure = 2afe  
luminous\_flux = 2aff  
luminous\_flux\_range = 2b00  
luminous\_intensity = 2b01  
magnetic\_declination = 2a2c  
magnetic\_flux\_density\_2d = 2aa0  
magnetic\_flux\_density\_3d = 2aa1  
manufacturer\_name\_string = 2a29  
mass\_flow = 2b02  
maximum\_recommended\_heart\_rate = 2a91  
measurement\_interval = 2a21  
media\_control\_point = 2ba4  
media\_control\_point\_opcodes\_supported = 2ba5  
media\_player\_icon\_object\_id = 2b94  
media\_player\_icon\_object\_type = 2ba9  
media\_player\_icon\_url = 2b95  
media\_player\_name = 2b93  
media\_state = 2ba3  
mesh\_provisioning\_data\_in = 2adb  
mesh\_provisioning\_data\_out = 2adc

```
mesh_proxy_data_in = 2add
mesh_proxy_data_out = 2ade
methane_concentration = 2bd1
middle_name = 2b48
model_number_string = 2a24
mute = 2bc3
navigation = 2a68
network_availability = 2a3e
new_alert = 2a46
next_track_object_id = 2b9e
nitrogen_dioxide_concentration = 2bd2
non_methane_volatile_organic_compounds_concentration = 2bd3
object_action_control_point = 2ac5
object_changed = 2ac8
object_first_created = 2ac1
object_id = 2ac3
object_last_modified = 2ac2
object_list_control_point = 2ac6
object_list_filter = 2ac7
object_name = 2abe
object_properties = 2ac4
object_size = 2ac0
object_type = 2abf
ots_feature = 2abd
ozone_concentration = 2bd4
parent_group_object_id = 2b9f
particulate_matter_10_concentration = 2bd7
particulate_matter_1_concentration = 2bd5
particulate_matter_2_5_concentration = 2bd6
perceived_lightness = 2b03
percentage_8 = 2b04
```

peripheral\_preferred\_connection\_parameters = 2a04  
peripheral\_privacy\_flag = 2a02  
physical\_activity\_monitor\_control\_point = 2b43  
physical\_activity\_monitor\_features = 2b3b  
physical\_activity\_session\_descriptor = 2b45  
playback\_speed = 2b9a  
playing\_order = 2ba1  
playing\_orders\_supported = 2ba2  
plx\_continuous\_measurement = 2a5f  
plx\_features = 2a60  
plx\_spot\_check\_measurement = 2a5e  
pnp\_id = 2a50  
pollen\_concentration = 2a75  
position\_2d = 2a2f  
position\_3d = 2a30  
position\_quality = 2a69  
power = 2b05  
power\_specification = 2b06  
preferred\_units = 2b46  
pressure = 2a6d  
protocol\_mode = 2a4e  
pulse\_oximetry\_control\_point = 2a62  
rainfall = 2a78  
rc\_feature = 2b1d  
rc\_settings = 2b1e  
reconnection\_address = 2a03  
reconnection\_configuration\_control\_point = 2b1f  
record\_access\_control\_point = 2a52  
reference\_time\_information = 2a14  
registered\_user\_characteristic = 2b37  
relative\_runtime\_current\_range = 2b07

relative\_runtime\_generic\_level\_range = 2b08  
relative\_value\_period\_of\_day = 2b0b  
relative\_value\_temperature\_range = 2b0c  
relative\_value\_voltage\_range = 2b09  
relative\_value\_illuminance\_range = 2b0a  
removable = 2a3a  
report = 2a4d  
report\_map = 2a4b  
resolvable\_private\_address\_only = 2ac9  
resting\_heart\_rate = 2a92  
ringer\_control\_point = 2a40  
ringer\_setting = 2a41  
rower\_data = 2ad1  
rsc\_feature = 2a54  
rsc\_measurement = 2a53  
sc\_control\_point = 2a55  
scan\_interval\_window = 2a4f  
scan\_refresh = 2a31  
scientific\_temperature\_celsius = 2a3c  
secondary\_time\_zone = 2a10  
search\_control\_point = 2ba7  
search\_results\_object\_id = 2ba6  
sedentary\_interval\_notification = 2b4f  
seeking\_speed = 2b9b  
sensor\_location = 2a5d  
serial\_number\_string = 2a25  
server\_supported\_features = 2b3a  
service\_changed = 2a05  
service\_required = 2a3b  
set\_identity\_resolving\_key = 2b84  
set\_member\_lock = 2b86



```
set_member_rank = 2b87
sink_ase = 2bc4
sink_audio_locations = 2bca
sink_pac = 2bc9
sleep_activity_instantaneous_data = 2b41
sleep_activity_summary_data = 2b42
software_revision_string = 2a28
source_ase = 2bc5
source_audio_locations = 2bcc
source_pac = 2bcb
sport_type_for_aerobic_and_anaerobic_thresholds = 2a93
stair_climber_data = 2ad0
status_flags = 2bbb
step_climber_data = 2acf
step_counter_activity_summary_data = 2b40
stride_length = 2b49
string = 2a3d
sulfur_dioxide_concentration = 2bd8
sulfur_hexafluoride_concentration = 2bd9
supported_audio_contexts = 2bce
supported_heart_rate_range = 2ad7
supported_inclination_range = 2ad5
supported_new_alert_category = 2a47
supported_power_range = 2ad8
supported_resistance_level_range = 2ad6
supported_speed_range = 2ad4
supported_unread_alert_category = 2a48
system_id = 2a23
tds_control_point = 2abc
temperature = 2a6e
temperature_celsius = 2a1f
```

```
temperature_fahrenheit = 2a20
temperature_8 = 2b0d
temperature_8_in_a_period_of_day = 2b0e
temperature_8_statistics = 2b0f
temperature_measurement = 2a1c
temperature_range = 2b10
temperature_statistics = 2b11
temperature_type = 2a1d
termination_reason = 2bc0
three_zone_heart_rate_limits = 2a94
time_accuracy = 2a12
time_broadcast = 2a15
time_change_log_data = 2b92
time_decihour_8 = 2b12
time_exponential_8 = 2b13
time_hour_24 = 2b14
time_millisecond_24 = 2b15
time_second_16 = 2b16
time_second_8 = 2b17
time_source = 2a13
time_update_control_point = 2a16
time_update_state = 2a17
time_with_dst = 2a11
time_zone = 2a0e
tmap_role = 2b51
track_changed = 2b96
track_duration = 2b98
track_object_type = 2bab
track_position = 2b99
track_segments_object_type = 2baa
track_title = 2b97
```

```
training_status = 2ad3
treadmill_data = 2acd
true_wind_direction = 2a71
true_wind_speed = 2a70
two_zone_heart_rate_limit = 2a95
tx_power_level = 2a07
uncertainty = 2ab4
unread_alert_status = 2a45
unspecified = 2aca
uri = 2ab6
user_control_point = 2a9f
user_index = 2a9a
uv_index = 2a76
vo2_max = 2a96
voltage = 2b18
voltage_specification = 2b19
voltage_statistics = 2b1a
volume_control_point = 2b7e
volume_flags = 2b7f
volume_flow = 2b1b
volume_offset_control_point = 2b82
volume_offset_state = 2b80
volume_state = 2b7d
waist_circumference = 2a97
weight = 2a98
weight_measurement = 2a9d
weight_scale_feature = 2a9e
wind_chill = 2a79
```

```

blatann.bt_sig.uuids.company_assigned_uuid16s = {fce1: 'Sony Group Corporation', fce2:
'Baracoda Daily Healthtech', fce3: 'Smith & Nephew Medical Limited', fce4: 'Samsara
Networks, Inc', fce5: 'Samsara Networks, Inc', fce6: 'Guard RFID Solutions Inc.', fce7:
'TKH Security B.V.', fce8: 'ITT Industries', fce9: 'MindRhythm, Inc.', fcea: 'Chess
Wise B.V.', fceb: 'Avi-On', fcec: 'Griffwerk GmbH', fced: 'Workaround GmbH', fcee:
'Velentium, LLC', fcef: 'Divesoft s.r.o.', fcf0: 'Security Enhancement Systems, LLC',
fcf1: 'Google LLC', fcf2: 'Bitwards Oy', fcf3: 'Armatura LLC', fcf4: 'Allegion',
fcf5: 'Trident Communication Technology, LLC', fcf6: 'The Linux Foundation', fcf7:
'Honor Device Co., Ltd.', fcf8: 'Honor Device Co., Ltd.', fcf9: 'Leupold & Stevens,
Inc.', fcfa: 'Leupold & Stevens, Inc.', fcfb: 'Shenzhen Benwei Media Co., Ltd.', fcfc:
'Barrot Technology Limited', fcfd: 'Barrot Technology Limited', fcfe: 'Sennheiser
Consumer Audio GmbH', fcff: '701x', fd00: 'FUTEK Advanced Sensor Technology, Inc.',
fd01: 'Sanvita Medical Corporation', fd02: 'LEGO System A/S', fd03: 'Quuppa Oy', fd04:
'Shure Inc.', fd05: 'Qualcomm Technologies, Inc.', fd06: 'RACE-AI LLC', fd07:
'Swedlock AB', fd08: 'Bull Group Incorporated Company', fd09: 'Cousins and Sears LLC',
fd0a: 'Luminostics, Inc.', fd0b: 'Luminostics, Inc.', fd0c: 'OSM HK Limited', fd0d:
'Blecon Ltd', fd0e: 'HerdDogg, Inc', fd0f: 'AEON MOTOR CO.,LTD.', fd10: 'AEON MOTOR
CO.,LTD.', fd11: 'AEON MOTOR CO.,LTD.', fd12: 'AEON MOTOR CO.,LTD.', fd13: 'BRG
Sports, Inc.', fd14: 'BRG Sports, Inc.', fd15: 'Panasonic Corporation', fd16:
'Sensitech, Inc.', fd17: 'LEGIC Identsystems AG', fd18: 'LEGIC Identsystems AG', fd19:
'Smith & Nephew Medical Limited', fd1a: 'CSIRO', fd1b: 'Helios Sports, Inc.', fd1c:
'Brady Worldwide Inc.', fd1d: 'Samsung Electronics Co., Ltd', fd1e: 'Plume Design
Inc.', fd1f: '3M', fd20: 'GN Hearing A/S', fd21: 'Huawei Technologies Co., Ltd.',
fd22: 'Huawei Technologies Co., Ltd.', fd23: 'DOM Sicherheitstechnik GmbH & Co. KG',
fd24: 'GD Midea Air-Conditioning Equipment Co., Ltd.', fd25: 'GD Midea Air-Conditioning
Equipment Co., Ltd.', fd26: 'Novo Nordisk A/S', fd27: 'i2Systems', fd28: 'Julius Blum
GmbH', fd29: 'Asahi Kasei Corporation', fd2a: 'Sony Corporation', fd2b: 'The Access
Technologies', fd2c: 'The Access Technologies', fd2d: 'Xiaomi Inc.', fd2e: 'Bitstrata
Systems Inc.', fd2f: 'Bitstrata Systems Inc.', fd30: 'Sesam Solutions BV', fd31: 'LG
Electronics Inc.', fd32: 'Gemalto Holding BV', fd33: 'DashLogic, Inc.', fd34:
'Aerosens LLC.', fd35: 'Transsion Holdings Limited', fd36: 'Google LLC', fd37:
'TireCheck GmbH', fd38: 'Danfoss A/S', fd39: 'PREDIKTAS', fd3a: 'Verkada Inc.', fd3b:
'Verkada Inc.', fd3c: 'Redline Communications Inc.', fd3d: 'Woan Technology (Shenzhen)
Co., Ltd.', fd3e: 'Pure Watercraft, inc.', fd3f: 'Cognosos, Inc', fd40: 'Beflex Inc.',
fd41: 'Amazon Lab126', fd42: 'Globe (Jiangsu) Co.,Ltd', fd43: 'Apple Inc.', fd44:
'Apple Inc.', fd45: 'GB Solution co.,Ltd', fd46: 'Lemco IKE', fd47: 'Liberty Global
Inc.', fd48: 'Geberit International AG', fd49: 'Panasonic Corporation', fd4a: 'Sigma
Elektro GmbH', fd4b: 'Samsung Electronics Co., Ltd.', fd4c: 'Adolf Wuerth GmbH & Co
KG', fd4d: '70mai Co.,Ltd.', fd4e: '70mai Co.,Ltd.', fd4f: 'Forkbeard Technologies
AS', fd50: 'Hangzhou Tuya Information Technology Co., Ltd', fd51: 'UTC Fire and
Security', fd52: 'UTC Fire and Security', fd53: 'PCI Private Limited', fd54: 'Qingdao
Haier Technology Co., Ltd.', fd55: 'Braveheart Wireless, Inc.', fd57: 'Volvo Car
Corporation', fd58: 'Volvo Car Corporation', fd59: 'Samsung Electronics Co., Ltd.',
fd5a: 'Samsung Electronics Co., Ltd.', fd5b: 'V2SOFT INC.', fd5c: 'React Mobile',
fd5d: 'maxon motor ltd.', fd5e: 'Tapkey GmbH', fd5f: 'Oculus VR, LLC', fd60: 'Sercomm
Corporation', fd61: 'Arendi AG', fd62: 'Fitbit, Inc.', fd63: 'Fitbit, Inc.', fd64:
'INRIA', fd65: 'Razer Inc.', fd66: 'Zebra Technologies Corporation', fd67: 'Montblanc
Simplo GmbH', fd68: 'Ubique Innovation AG', fd69: 'Samsung Electronics Co., Ltd.',
fd6a: 'Emerson', fd6b: 'rapitag GmbH', fd6c: 'Samsung Electronics Co., Ltd.', fd6d:
'Sigma Elektro GmbH', fd6e: 'Polidea sp. z o.o.', fd6f: 'Apple, Inc.', fd70:
'GuangDong Oppo Mobile Telecommunications Corp., Ltd.', fd71: 'GN Hearing A/S', fd72:
'Logitech International SA', fd73: 'BRControls Products BV', fd74: 'BRControls Products
BV', fd75: 'Insulet Corporation', fd76: 'Insulet Corporation', fd77: 'Withings', fd78:
'Withings', fd79: 'Withings', fd7a: 'Withings', fd7b: 'WYZE LABS, INC.', fd7c:
'Toshiba Information Systems(Japan) Corporation', fd7d: 'Center for Advanced Research
Wernher Von Braun', fd7e: 'Samsung Electronics Co., Ltd.', fd7f: 'Husqvarna AB', fd80:
'Phindex Technologies, Inc', fd81: 'CANDY HOUSE, Inc.', fd82: 'Shy Corporation', fd83:
'iNFORM Technology GmbH', fd84: 'Tile, Inc.', fd85: 'Husqvarna AB', fd86: 'Abbott',
fd87: 'Google LLC', fd88: 'Urbanminded LTD', fd89: 'Urbanminded LTD', fd8a: 'Signify
Netherlands B.V.', fd8b: 'Jigowatts Inc.', fd8c: 'Google LLC', fd8d: 'quip NYC Inc.',

```

16-bit UUIDs assigned to companies by Bluetooth SIG

`blatann.bt_sig.uuids.t`

alias of *CharacteristicUuid*

## blatann.examples package

### Submodules

#### blatann.examples.broadcaster module

This is an example of a broadcaster role BLE device. It advertises as a non-connectable device and emits the device's current time as a part of the advertising data.

`blatann.examples.broadcaster.wait_for_user_stop(stop_event)`

`blatann.examples.broadcaster.main(serial_port)`

#### blatann.examples.central\_uart\_service module

This example implements Nordic's custom UART service and demonstrates how to configure the MTU size. It is configured to use an MTU size based on the Data Length Extensions feature of BLE for maximum throughput. This is compatible with the `peripheral_uart_service` example.

This is a simple example which just echos back any data that the client sends to it.

`blatann.examples.central_uart_service.on_connect(peer, event_args)`

Event callback for when a central device connects to us

##### Parameters

- **peer** (`blatann.peer.Client`) – The peer that connected to us
- **event\_args** – None

`blatann.examples.central_uart_service.on_disconnect(peer, event_args)`

Event callback for when the client disconnects from us (or when we disconnect from the client)

##### Parameters

- **peer** (`blatann.peer.Client`) – The peer that disconnected
- **event\_args** (`blatann.event_args.DisconnectionEventArgs`) – The event args

`blatann.examples.central_uart_service.on_mtu_size_update(peer, event_args)`

Callback for when the peer's MTU size has been updated/negotiated

##### Parameters

- **peer** (`blatann.peer.Client`) – The peer the MTU was updated on

`blatann.examples.central_uart_service.on_data_rx(service, data)`

Called whenever data is received on the RX line of the Nordic UART Service

##### Parameters

- **service** (`nordic_uart.service.NordicUartClient`) – the service the data was received from

- **data** (*bytes*) – The data that was received

```
blatann.examples.central_uart_service.main(serial_port)
```

### blatann.examples.central module

This example demonstrates implementing a central BLE connection in a procedural manner. Each bluetooth operation performed is done sequentially in a linear fashion, and the main context blocks until each operation completes before moving on to the rest of the program

This is designed to work alongside the peripheral example running on a separate nordic chip

```
blatann.examples.central.on_counting_char_notification(characteristic, event_args)
```

Callback for when a notification is received from the peripheral's counting characteristic. The peripheral will periodically notify a monotonically increasing, 4-byte integer. This callback unpacks the value and logs it out

#### Parameters

- **characteristic** (`blatann.gatt.gattc.GattcCharacteristic`) – The characteristic the notification was on (counting characteristic)
- **event\_args** (`blatann.event_args.NotificationReceivedEventArgs`) – The event arguments

```
blatann.examples.central.on_passkey_entry(peer, passkey_event_args)
```

Callback for when the user is requested to enter a passkey to resume the pairing process. Requests the user to enter the passkey and resolves the event with the passkey entered

#### Parameters

- **peer** – the peer the passkey is for
- **passkey\_event\_args** (`blatann.event_args.PasskeyEntryEventArgs`) –

```
blatann.examples.central.on_peripheral_security_request(peer, event_args)
```

Handler for peripheral-initiated security requests. This is useful in the case that the application wants to override the default response to peripheral-initiated security requests based on parameters, the peer, etc.

For example, to reject new pairing requests but allow already-bonded devices to enable encryption, one could use the `event_args.is_bonded_device` flag to accept or reject the request.

This handler is optional. If not provided the `SecurityParameters.reject_pairing_requests` parameter will determine the action to take.

#### Parameters

- **peer** (`blatann.peer.Peer`) – The peer that requested security
- **event\_args** (`blatann.event_args.PeripheralSecurityRequestEventArgs`) – The event arguments

```
blatann.examples.central.main(serial_port)
```

### blatann.examples.central\_battery\_service module

This example demonstrates reading and subscribing to a peripheral's Battery service to get updates on the peripheral's current battery levels. The operations here are programmed in a procedural manner.

This can be used alongside any peripheral which implements the Battery Service and advertises the 16-bit Battery Service UUID. The `peripheral_battery_service` example can be used with this.

```
blatann.examples.central_battery_service.on_battery_level_update(battery_service, event_args)
```

#### Parameters

**battery\_service** –

```
blatann.examples.central_battery_service.main(serial_port)
```

### blatann.examples.central\_descriptors module

This example shows how to read descriptors of a peripheral's characteristic.

This can be used with the `peripheral_descriptor` example running on a separate nordic device.

```
blatann.examples.central_descriptors.main(serial_port)
```

### blatann.examples.central\_device\_info\_service module

This example demonstrates reading a peripheral's Device Info Service using blatann's `device_info` service module. The operations here are programmed in a procedural manner.

This can be used alongside any peripheral which implements the DIS and advertises the 16-bit DIS service UUID. The `peripheral_device_info_service` example can be used with this.

```
blatann.examples.central_device_info_service.main(serial_port)
```

### blatann.examples.central\_event\_driven module

This example demonstrates programming with blatann in an event-driven, object-oriented manner. The BLE device is set up in the main context, however all logic past that point is done using event callbacks. The main context is blocked by a "GenericWaitable", which is notified when the program completes its intended function.

The program's logic itself is equivalent to the `central` example, where it connects and pairs to a device, registers a notification callback for the counting characteristic, then tests out the conversion of strings to hex.

One thing to note: when using event-driven callbacks, it is imperative that the callbacks themselves do not ever block on events (i.e. use the `.wait()` functionality). If this happens, you are essentially blocking the event thread from processing any more events and will wait indefinitely. A good rule of thumb when using blatann is just to not mix blocking and non-blocking calls.

This is designed to work alongside the `peripheral` example running on a separate nordic chip

```
class blatann.examples.central_event_driven.HexConverterTest(characteristic, waitable)
```

Bases: `object`

Class to perform the hex conversion process. It is passed in the hex conversion characteristic and the waitable to signal when the process completes

**start()**

Starts a new hex conversion process by writing the data to the peripheral's characteristic

**class** `blatann.examples.central_event_driven.MyPeripheralConnection`(*peer, waitable*)

Bases: `object`

Class to handle the post-connection database discovery and pairing process

**class** `blatann.examples.central_event_driven.ConnectionManager`(*ble\_device, exit\_waitable*)

Bases: `object`

Manages scanning and connecting to the target peripheral

**scan\_and\_connect**(*name, timeout=4*)

Starts the scanning process and sets up the callback for when scanning completes

**Parameters**

- **name** – The name of the peripheral to look for
- **timeout** – How long to scan for

`blatann.examples.central_event_driven.main`(*serial\_port*)

**blatann.examples.constants module****blatann.examples.example\_utils module**

`blatann.examples.example_utils.find_target_device`(*ble\_device, name*)

Starts the scanner and searches the advertising report for the desired name. If found, returns the peer's address that can be connected to

**Parameters**

- **ble\_device** (`blatann.BleDevice`) – The ble device to operate on
- **name** – The device's local name that is advertised

**Returns**

The peer's address if found, or None if not found

**blatann.examples.peripheral module**

This example exhibits some of the functionality of a peripheral BLE device, such as reading, writing and notifying characteristics.

This peripheral can be used with one of the central examples running on a separate nordic device, or can be run with the nRF Connect app to explore the contents of the service

`blatann.examples.peripheral.on_connect`(*peer, event\_args*)

Event callback for when a central device connects to us

**Parameters**

- **peer** (`blatann.peer.Client`) – The peer that connected to us
- **event\_args** – None



`blatann.examples.peripheral.on_disconnect(peer, event_args)`

Event callback for when the client disconnects from us (or when we disconnect from the client)

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that disconnected
- **event\_args** (`blatann.event_args.DisconnectionEventArgs`) – The event args

`blatann.examples.peripheral.on_hex_conversion_characteristic_write(characteristic, event_args)`

Event callback for when the client writes to the hex conversion characteristic. This takes the data written, converts it to the hex representation, and updates the characteristic with this new value. If the client is subscribed to the characteristic, the client will be notified.

#### Parameters

- **characteristic** (`blatann.gatt.gatts.GattsCharacteristic`) – The hex conversion characteristic
- **event\_args** (`blatann.event_args.WriteEventArgs`) – the event arguments

`blatann.examples.peripheral.on_gatts_subscription_state_changed(characteristic, event_args)`

Event callback for when a client subscribes or unsubscribes from a characteristic. This is the equivalent to when a client writes to a CCCD descriptor on a characteristic.

`blatann.examples.peripheral.on_time_char_read(characteristic, event_args)`

Event callback for when the client reads our time characteristic. Gets the current time and updates the characteristic. This demonstrates “lazy evaluation” of characteristics—instead of having to constantly update this characteristic, it is only updated when read/observed by an outside actor.

#### Parameters

- **characteristic** (`blatann.gatt.gatts.GattsCharacteristic`) – the time characteristic
- **event\_args** – None

`blatann.examples.peripheral.on_discovery_complete(peer, event_args)`

Callback for when the service discovery completes on the client. This will look for the client’s Device name characteristic (part of the Generic Access Service) and read the value

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer the discovery completed on
- **event\_args** (`blatann.event_args.DatabaseDiscoveryCompleteEventArgs`) – The event arguments (unused)

`blatann.examples.peripheral.on_security_level_changed(peer, event_args)`

Called when the security level changes, i.e. a bonded device connects and enables encryption or pairing has finished. If security has been enabled (i.e. bonded) and the peer’s services have yet to be discovered, discover now.

This code demonstrates that even in a peripheral connection role, the peripheral can still discover the database on the client, if the client has a database.

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that security was changed to
- **event\_args** (`blatann.event_args.SecurityLevelChangedEventArgs`) – the event arguments

`blatann.examples.peripheral.on_client_pairing_complete(peer, event_args)`

Event callback for when the pairing process completes with the client

**Parameters**

- **peer** (`blatann.peer.Client`) – the peer that completed pairing
- **event\_args** (`blatann.event_args.PairingCompleteEventArgs`) – the event arguments

`blatann.examples.peripheral.on_passkey_display(peer, event_args)`

Event callback that is called when a passkey is required to be displayed to a user for the pairing process.

**Parameters**

- **peer** (`blatann.peer.Client`) – The peer the passkey is for
- **event\_args** (`blatann.event_args.PasskeyDisplayEventArgs`) – The event args

`blatann.examples.peripheral.on_passkey_entry(peer, passkey_event_args)`

Callback for when the user is requested to enter a passkey to resume the pairing process. Requests the user to enter the passkey and resolves the event with the passkey entered

**Parameters**

- **peer** – the peer the passkey is for
- **passkey\_event\_args** (`blatann.event_args.PasskeyEntryEventArgs`) –

`class blatann.examples.peripheral.CountingCharacteristicThread(characteristic)`

Bases: `object`

Thread which updates the counting characteristic and notifies the client each time its updated. This also demonstrates the notification queuing functionality—if a notification/indication is already in progress, future notifications will be queued and sent out when the previous ones complete.

**join()**

Used to stop and join the thread

**run()**

`blatann.examples.peripheral.on_conn_params_updated(peer, event_args)`

`blatann.examples.peripheral.main(serial_port)`

## **blatann.examples.peripheral\_battery\_service module**

This example demonstrates using Bluetooth SIG's Battery service as a peripheral. The peripheral adds the service, then updates the battery level percentage periodically.

This can be used in conjunction with the nRF Connect apps to explore the functionality demonstrated

`blatann.examples.peripheral_battery_service.on_connect(peer, event_args)`

Event callback for when a central device connects to us

**Parameters**

- **peer** (`blatann.peer.Client`) – The peer that connected to us
- **event\_args** – None

`blatann.examples.peripheral_battery_service.on_disconnect(peer, event_args)`

Event callback for when the client disconnects from us (or when we disconnect from the client)

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that disconnected
- **event\_args** (`blatann.event_args.DisconnectionEventArgs`) – The event args

`blatann.examples.peripheral_battery_service.main(serial_port)`

### **blatann.examples.peripheral\_current\_time\_service module**

This example demonstrates using Bluetooth SIG’s defined Current Time service as a peripheral.

`blatann.examples.peripheral_current_time_service.on_connect(peer, event_args)`

Event callback for when a central device connects to us

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that connected to us
- **event\_args** – None

`blatann.examples.peripheral_current_time_service.on_disconnect(peer, event_args)`

Event callback for when the client disconnects from us (or when we disconnect from the client)

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that disconnected
- **event\_args** (`blatann.event_args.DisconnectionEventArgs`) – The event args

`blatann.examples.peripheral_current_time_service.on_current_time_write(characteristic, event_args)`

Callback registered to be triggered whenever the Current Time characteristic is written to

#### Parameters

- **characteristic** –
- **event\_args** (`blatann.event_args.DecodedWriteEventArgs`) – The write event args

`blatann.examples.peripheral_current_time_service.on_local_time_info_write(characteristic, event_args)`

Callback registered to be triggered whenever the Local Time Info characteristic is written to

#### Parameters

- **characteristic** –
- **event\_args** (`blatann.event_args.DecodedWriteEventArgs`) – The write event args

`blatann.examples.peripheral_current_time_service.main(serial_port)`

### blatann.examples.peripheral\_descriptors module

This example shows how to add descriptors to a characteristic in a GATT Database.

This can be used with the `central_descriptor` example running on a separate nordic device or can be run with the nRF Connect app

`blatann.examples.peripheral_descriptors.on_connect(peer, event_args)`

Event callback for when a central device connects to us

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that connected to us
- **event\_args** – None

`blatann.examples.peripheral_descriptors.on_disconnect(peer, event_args)`

Event callback for when the client disconnects from us (or when we disconnect from the client)

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that disconnected
- **event\_args** (`blatann.event_args.DisconnectionEventArgs`) – The event args

`blatann.examples.peripheral_descriptors.on_read(characteristic, event_args)`

On read for the Time characteristic. Updates the characteristic with the current UTC time as a 32-bit number

`blatann.examples.peripheral_descriptors.main(serial_port)`

### blatann.examples.peripheral\_device\_info\_service module

This example shows how to implement a Device Info Service on a peripheral.

This example can be used alongside the `central_device_info_service` example running on another nordic device, or using the Nordic nRF Connect app to connect and browse the peripheral's service data

`blatann.examples.peripheral_device_info_service.on_connect(peer, event_args)`

Event callback for when a central device connects to us

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that connected to us
- **event\_args** – None

`blatann.examples.peripheral_device_info_service.on_disconnect(peer, event_args)`

Event callback for when the client disconnects from us (or when we disconnect from the client)

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that disconnected
- **event\_args** (`blatann.event_args.DisconnectionEventArgs`) – The event args

`blatann.examples.peripheral_device_info_service.main(serial_port)`

**blatann.examples.peripheral\_glucose\_service module**

This example demonstrates using Bluetooth SIG's defined Glucose service as a peripheral. The peripheral creates a range of fake glucose readings that can be queried from the central.

This can be used in conjunction with the nRF Connect apps to explore the peripheral's functionality

`blatann.examples.peripheral_glucose_service.on_connect(peer, event_args)`

Event callback for when a central device connects to us

**Parameters**

- **peer** (`blatann.peer.Client`) – The peer that connected to us
- **event\_args** – None

`blatann.examples.peripheral_glucose_service.on_disconnect(peer, event_args)`

Event callback for when the client disconnects from us (or when we disconnect from the client)

**Parameters**

- **peer** (`blatann.peer.Client`) – The peer that disconnected
- **event\_args** (`blatann.event_args.DisconnectionEventArgs`) – The event args

`blatann.examples.peripheral_glucose_service.on_security_level_changed(peer, event_args)`

Event callback for when the security level changes on a connection with a peer

**Parameters**

- **peer** (`blatann.peer.Client`) – The peer the security level changed on
- **event\_args** (`blatann.event_args.SecurityLevelChangedEventArgs`) – The event args

`blatann.examples.peripheral_glucose_service.display_passkey(peer, event_args)`

Event callback that is called when a passkey is required to be displayed to a user for the pairing process.

**Parameters**

- **peer** (`blatann.peer.Client`) – The peer the passkey is for
- **event\_args** (`blatann.event_args.PasskeyDisplayEventArgs`) – The event args

`blatann.examples.peripheral_glucose_service.add_fake_glucose_readings(glucose_database, num_records=15)`

Helper method to create some glucose readings and add them to the glucose database

**Parameters**

- **glucose\_database** (`glucose.BasicGlucoseDatabase`) – The database to add readings to
- **num\_records** – The number of records to generate

`blatann.examples.peripheral_glucose_service.main(serial_port)`

### blatann.examples.peripheral\_rssi module

This is a simple example which demonstrates enabling RSSI updates for active connections.

`blatann.examples.peripheral_rssi.on_rssi_changed(peer, rssi)`

Event callback for when the RSSI with the central device changes by the configured dBm threshold

#### Parameters

- **peer** – The peer object
- **rssi** (`int`) – The new RSSI for the connection

`blatann.examples.peripheral_rssi.on_connect(peer, event_args)`

Event callback for when a central device connects to us

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that connected to us
- **event\_args** – None

`blatann.examples.peripheral_rssi.on_disconnect(peer, event_args)`

Event callback for when the client disconnects from us (or when we disconnect from the client)

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that disconnected
- **event\_args** (`blatann.event_args.DisconnectionEventArgs`) – The event args

`blatann.examples.peripheral_rssi.main(serial_port)`

### blatann.examples.peripheral\_uart\_service module

This example implements Nordic’s custom UART service and demonstrates how to configure the MTU size. It is configured to use an MTU size based on the Data Length Extensions feature of BLE for maximum throughput. This is compatible with the nRF Connect app (Android version tested) and the `central_uart_service` example.

This is a simple example which just echos back any data that the client sends to it.

`blatann.examples.peripheral_uart_service.on_connect(peer, event_args)`

Event callback for when a central device connects to us

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that connected to us
- **event\_args** – None

`blatann.examples.peripheral_uart_service.on_disconnect(peer, event_args)`

Event callback for when the client disconnects from us (or when we disconnect from the client)

#### Parameters

- **peer** (`blatann.peer.Client`) – The peer that disconnected
- **event\_args** (`blatann.event_args.DisconnectionEventArgs`) – The event args

`blatann.examples.peripheral_uart_service.on_mtu_size_update(peer, event_args)`

Callback for when the peer's MTU size has been updated/negotiated

**Parameters**

**peer** (`blatann.peer.Client`) – The peer the MTU was updated on

`blatann.examples.peripheral_uart_service.on_data_rx(service, data)`

Called whenever data is received on the RX line of the Nordic UART Service

**Parameters**

- **service** (`nordic_uart.service.NordicUartServer`) – the service the data was received from
- **data** (`bytes`) – The data that was received

`blatann.examples.peripheral_uart_service.on_tx_complete(service, event_args)`

`blatann.examples.peripheral_uart_service.main(serial_port)`

## **blatann.examples.scanner module**

This example simply demonstrates scanning for peripheral devices

`blatann.examples.scanner.main(serial_port)`

## **blatann.gap package**

`blatann.gap.HciStatus`

The default link-layer packet size used when a connection is established

`blatann.gap.DLE_SIZE_DEFAULT = 27`

The minimum allowed link-layer packet size

`blatann.gap.DLE_SIZE_MINIMUM = 27`

The maximum allowed link-layer packet size

## **Submodules**

### **blatann.gap.advertise\_data module**

`class blatann.gap.advertise_data.AdvertisingFlags`

Bases: `object`

`LIMITED_DISCOVERY_MODE = 1`

`GENERAL_DISCOVERY_MODE = 2`

`BR_EDR_NOT_SUPPORTED = 4`

`BR_EDR_CONTROLLER = 8`

`BR_EDR_HOST = 16`

```
class blatann.gap.advertise_data.AdvertisingData(flags=None, local_name=None,  
                                                  local_name_complete=True,  
                                                  service_uuid16s=None, service_uuid128s=None,  
                                                  has_more_uuid16_services=False,  
                                                  has_more_uuid128_services=False,  
                                                  service_data=None, manufacturer_data=None,  
                                                  **other_entries)
```

Bases: `object`

Class which represents data that can be advertised

**MAX\_ENCODED\_LENGTH** = 31

```
class Types(value)
```

Bases: `Enum`

An enumeration.

**flags** = 1

**service\_16bit\_uuid\_more\_available** = 2

**service\_16bit\_uuid\_complete** = 3

**service\_32bit\_uuid\_more\_available** = 4

**service\_32bit\_uuid\_complete** = 5

**service\_128bit\_uuid\_more\_available** = 6

**service\_128bit\_uuid\_complete** = 7

**short\_local\_name** = 8

**complete\_local\_name** = 9

**tx\_power\_level** = 10

**class\_of\_device** = 13

**simple\_pairing\_hash\_c** = 14

**simple\_pairing\_randimizer\_r** = 15

**security\_manager\_tk\_value** = 16

**security\_manager\_oob\_flags** = 17

**slave\_connection\_interval\_range** = 18

**solicited\_service\_uuids\_16bit** = 20

**solicited\_service\_uuids\_128bit** = 21

**service\_data** = 22

**public\_target\_address** = 23

**random\_target\_address** = 24



```

appearance = 25
advertising_interval = 26
le_bluetooth_device_address = 27
le_role = 28
simple_pairng_hash_c256 = 29
simple_pairng_randomizer_r256 = 30
service_data_32bit_uuid = 32
service_data_128bit_uuid = 33
uri = 36
information_3d_data = 61
manufacturer_specific_data = 255

```

**property flags:** `Optional[int]`

The advertising flags in the payload, if set

**Getter**

Gets the advertising flags in the payload, or None if not set

**Setter**

Sets the advertising flags in the payload

**Delete**

Removes the advertising flags from the payload

**property service\_data:** `Optional[Union[bytes, List[int]]]`

The service data in the payload, if set

**Getter**

Gets the service data in the payload, or None if not set

**Setter**

Sets the service data for the payload

**Delete**

Removes the service data from the payload

**property manufacturer\_data:** `Optional[Union[bytes, List[int]]]`

The manufacturer data in the payload, if set

**Getter**

Gets the manufacturer data in the payload, or None if not set

**Setter**

Sets the manufacturer data for the payload

**Delete**

Removes the manufacturer data from the payload

**property service\_uuids:** `List[Uuid]`

Gets all of the 16-bit and 128-bit service UUIDs specified in the advertising data

**check\_encoded\_length()**

Checks if the encoded length of this advertising data payload meets the maximum allowed length specified by the Bluetooth specification

**Return type**

`Tuple[int, bool]`

**Returns**

a tuple of the encoded length and a bool result of whether or not it meets requirements

**to\_ble\_adv\_data()**

Converts the advertising data to a BLEAdvData object that can be used by the nRF Driver

**Returns**

the BLEAdvData object which represents this advertising data

**Return type**

`nrf_types.BLEAdvData`

**to\_bytes()**

Converts the advertising data to the encoded bytes that will be advertised over the air. Advertising payloads are encoded in a length-type-value format

**Return type**

`bytes`

**Returns**

The encoded payload

**classmethod from\_ble\_adv\_records(*advertise\_records*)**

Converts a dictionary of AdvertisingData.Type: value keypairs into an object of this class

**Parameters**

**advertise\_records** (*dict*) – a dictionary mapping the advertise data types to their corresponding values

**Returns**

the AdvertisingData from the records given

**Return type**

*AdvertisingData*

**class** `blatann.gap.advertise_data.ScanReport`(*adv\_report*, *resolved\_address*)

Bases: `object`

Represents a payload and associated metadata that's received during scanning

**property** `device_name`: `str`

**Read Only**

The name of the device, pulled from the advertising data (if advertised) or uses the Peer's MAC Address if not set

**property** `is_bonded_device`: `bool`

If the scan report is from a BLE device that the local device has a matching bond database entry

**property** `resolved_address`: `Optional[PeerAddress]`

If the scan report is from a bonded device, this is the resolved public/static/random BLE address. This may be the same as `peer_addr` if the device is not advertising as a private resolvable address

**update**(*adv\_report*)

Used internally to merge a new advertising payload that was received into the current scan report

**class** `blatann.gap.advertise_data.ScanReportCollection`

Bases: `object`

Collection of all the advertising data and scan reports found in a scanning session

**property** `advertising_peers_found`: `Iterable[ScanReport]`

Gets the list of scans which have been combined and condensed into a list where each entry is a unique peer. The scan reports in this list represent aggregated data of each advertising packet received by the advertising device, such that later advertising packets will update/overwrite packet attributes received from earlier packets, if the data has been modified.

**Returns**

The list of scan reports, with each being a unique peer

**property** `all_scan_reports`: `Iterable[ScanReport]`

Gets the list of all of the individual advertising packets received.

**Returns**

The list of all scan reports

**get\_report\_for\_peer**(*peer\_addr*)

Gets the combined/aggregated scan report for a given Peer's address. If the peer's scan report isn't found, returns None

**Parameters**

**peer\_addr** – The peer's address to search for

**Return type**

`Optional[ScanReport]`

**Returns**

The associated scan report, if found

**clear**()

Clears out all of the scan reports cached

**update**(*adv\_report*, *resolved\_peer\_addr=None*)

Used internally to update the collection with a new advertising report received

**Return type**

`ScanReport`

**Returns**

The Scan Report created from the advertising report

**blatann.gap.advertising module****class** `blatann.gap.advertising.Advertiser`(*ble\_device*, *client*, *conn\_tag*=0)Bases: `object`

Class which manages the advertising state of the BLE Device

**ADVERTISE\_FOREVER** = 0

Special value used to indicate that the BLE device should advertise indefinitely until either a central is connected or stopped manually.

**property** `on_advertising_timeout`: `Event[Advertiser, None]`

Event generated whenever advertising times out and finishes with no connections made

---

**Note:** If auto-restart advertising is enabled, this will trigger on each advertising timeout configured

---

**Returns**

an Event which can have handlers registered to and deregistered from

**property** `is_advertising`: `bool`**Read Only**

Current state of advertising

**property** `min_interval_ms`: `float`**Read Only**

The minimum allowed advertising interval, in milliseconds. This is defined by the Bluetooth specification.

**property** `max_interval_ms`: `float`**Read Only**

The maximum allowed advertising interval, in milliseconds. This is defined by the Bluetooth specification.

**property** `auto_restart`: `bool`

Enables/disables whether or not the device should automatically restart advertising when an advertising timeout occurs or the client is disconnected.

---

**Note:** Auto-restart is disabled automatically when `stop()` is called

---

**Getter**

Gets the auto-restart flag

**Setter**

Sets/clears the auto-restart flag

**set\_channel\_mask**(*ch37\_enabled*=True, *ch38\_enabled*=True, *ch39\_enabled*=True)

Enables/disables which channels advertising packets are sent out on. By default, all 3 channels (37, 38, 39) are enabled. At least one of the 3 channels **MUST** be enabled, otherwise a `ValueError` exception will be raised.

This mask will take effect the next time advertising is started or restarted due to timeout/disconnect.

**Parameters**

- **ch37\_enabled** – True to enable advertising on channel 37, False to disable
- **ch38\_enabled** – True to enable advertising on channel 38, False to disable
- **ch39\_enabled** – True to enable advertising on channel 39, False to disable

**set\_advertise\_data**(*advertise\_data=AdvertisingData()*, *scan\_response=AdvertisingData()*)

Sets the advertising and scan response data which will be broadcasted to peers during advertising

---

**Note:** BLE Restricts advertise and scan response data to an encoded length of 31 bytes each. Use `AdvertisingData.check_encoded_length()` to determine if the payload is too large

---

#### Parameters

- **advertise\_data** (*AdvertisingData*) – The advertising data to use
- **scan\_response** (*AdvertisingData*) – The scan response data to use. This data is only sent when a scanning device requests the scan response packet (active scanning)

#### Raises

`InvalidOperationExpection` if one of the payloads is too large

**set\_default\_advertise\_params**(*advertise\_interval\_ms*, *timeout\_seconds*,  
*advertise\_mode=BLEGapAdvType.connectable\_undirected*)

Sets the default advertising parameters so they do not need to be specified on each start

#### Parameters

- **advertise\_interval\_ms** (*float*) – The advertising interval, in milliseconds. Should be a multiple of 0.625ms, otherwise it'll be rounded down to the nearest 0.625ms
- **timeout\_seconds** (*int*) – How long to advertise for before timing out, in seconds. For no timeout, use `ADVERTISE_FOREVER` (0)
- **advertise\_mode** (*BLEGapAdvType*) – The mode the advertiser should use

**start**(*adv\_interval\_ms=None*, *timeout\_sec=None*, *auto\_restart=None*, *advertise\_mode=None*)

Starts advertising with the given parameters. If none given, will use the default set through `set_default_advertise_params()`

#### Parameters

- **adv\_interval\_ms** (*Optional[float]*) – The interval at which to send out advertise packets, in milliseconds. Should be a multiple of 0.625ms, otherwise it'll be round down to the nearest 0.625ms
- **timeout\_sec** (*Optional[int]*) – The duration which to advertise for. For no timeout, use `ADVERTISE_FOREVER` (0)
- **auto\_restart** (*Optional[bool]*) – Flag indicating that advertising should restart automatically when the timeout expires, or when the client disconnects
- **advertise\_mode** (*Optional[BLEGapAdvType]*) – The mode the advertiser should use

#### Returns

A waitable that will expire either when the timeout occurs or a client connects. The waitable will return either `None` on timeout or `Client` on successful connection

#### Return type

*ClientConnectionWaitable*

**stop()**

Stops advertising and disables the auto-restart functionality (if enabled)

### **blatann.gap.bond\_db module**

**class** `blatann.gap.bond_db.BondingData(own_ltk, peer_ltk, peer_id, peer_sign)`

Bases: `object`

**classmethod** `from_keyset(bonding_keyset)`

**to\_dict()**

**classmethod** `from_dict(data)`

**class** `blatann.gap.bond_db.BondDbEntry(entry_id=0)`

Bases: `object`

**resolved\_peer\_address()**

Return type

`PeerAddress`

**matches\_peer(own\_address, peer\_address, peer\_is\_client, master\_id=None)**

Return type

`bool`

**peer\_address\_matches\_or\_resolves(peer\_address)**

Return type

`bool`

**to\_dict()**

**classmethod** `from_dict(data)`

**class** `blatann.gap.bond_db.BondDatabase`

Bases: `object`

**create()**

Return type

`BondDbEntry`

**add(db\_entry)**

**update(db\_entry)**

**delete(db\_entry)**

**delete\_all()**

**find\_entry(own\_address, peer\_address, peer\_is\_client, master\_id=None)**

**class** `blatann.gap.bond_db.BondDatabaseLoader`

Bases: `object`

**load()**

**Return type**

*BondDatabase*

**save(*db*)**

## blatann.gap.default\_bond\_db module

**class** blatann.gap.default\_bond\_db.DatabaseStrategy

Bases: *object*

Abstract base class defining the methods and properties for serializing/deserializing bond databases into different formats

**property file\_extension:** *str*

The file extension that this strategy can serialize/deserialize

**load(*filename*)**

Loads/deserializes a database file

**Parameters**

**filename** (*str*) – Name of the file to deserialize

**Return type**

*DefaultBondDatabase*

**Returns**

The loaded bond database

**save(*filename*, *db*)**

Saves/serializes a database to a file

**Parameters**

- **filename** (*str*) – Filename to save the database to
- **db** (*DefaultBondDatabase*) – The database object serialize

**class** blatann.gap.default\_bond\_db.JsonDatabaseStrategy

Bases: *DatabaseStrategy*

Strategy for serializing/deserializing bond databases in JSON format

**property file\_extension:** *str*

The file extension that this strategy can serialize/deserialize

**load(*filename*)**

Loads/deserializes a database file

**Parameters**

**filename** – Name of the file to deserialize

**Return type**

*DefaultBondDatabase*

**Returns**

The loaded bond database

**save**(*filename*, *db*)

Saves/serializes a database to a file

**Parameters**

- **filename** (*str*) – Filename to save the database to
- **db** (*DefaultBondDatabase*) – The database object serialize

**class** `blatann.gap.default_bond_db.PickleDatabaseStrategy`

Bases: *DatabaseStrategy*

Strategy for serializing/deserializing bond databases in pickle format

**property** `file_extension`: *str*

The file extension that this strategy can serialize/deserialize

**load**(*filename*)

Loads/deserializes a database file

**Parameters**

**filename** – Name of the file to deserialize

**Return type**

*DefaultBondDatabase*

**Returns**

The loaded bond database

**save**(*filename*, *db*)

Saves/serializes a database to a file

**Parameters**

- **filename** – Filename to save the database to
- **db** (*DefaultBondDatabase*) – The database object serialize

```
blatann.gap.default_bond_db.database_strategies =  
[<blatann.gap.default_bond_db.PickleDatabaseStrategy object>,  
<blatann.gap.default_bond_db.JsonDatabaseStrategy object>]
```

List of supported database strategies

```
blatann.gap.default_bond_db.database_strategies_by_extension: Dict[str,  
DatabaseStrategy] = {'json': <blatann.gap.default_bond_db.JsonDatabaseStrategy object>,  
'pkl': <blatann.gap.default_bond_db.PickleDatabaseStrategy object>}
```

Mapping of database file extensions to their respective strategies

**class** `blatann.gap.default_bond_db.DefaultBondDatabaseLoader`(*filename*='user')

Bases: *BondDatabaseLoader*

**migrate\_to\_json**(*base\_filename*)

**load**()

**Return type**

*DefaultBondDatabase*

**save**(*db*)

**class** `blatann.gap.default_bond_db.DefaultBondDatabase`(*records*=None)

Bases: *BondDatabase*



**create()**

**add**(*db\_entry*)

**update**(*db\_entry*)

**delete**(*db\_entry*)

**delete\_all()**

**find\_entry**(*own\_address, peer\_address, peer\_is\_client, master\_id=None*)

Attempts to find a bond entry which satisfies the parameters provided

#### Parameters

- **own\_address** (*PeerAddress*) – The local device’s BLE address
- **peer\_address** (*PeerAddress*) – The peer’s BLE address
- **peer\_is\_client** (*bool*) – Flag indicating the role of the peer. True if the peer is a client/central, False if the peer is a server/peripheral
- **master\_id** (*Optional[BLEGapMasterId]*) – If during a security info request, this is the Master ID provided by the peer to search for

#### Return type

*Optional[BondDbEntry]*

#### Returns

The first entry that satisfies the above parameters, or None if no entry was found

**blatann.gap.default\_bond\_db.migrate\_bond\_database**(*from\_file, to\_file*)

Migrates a bond database file from one format to another.

For supported extensions/formats, check `database_strategies_by_extension.keys()`

#### Parameters

- **from\_file** (*str*) – File to migrate from
- **to\_file** (*str*) – File to migrate to

## blatann.gap.gap\_types module

**class** `blatann.gap.gap_types.Phy`(*value*)

Bases: `IntFlag`

The supported PHYs

---

**Note:** Coded PHY is currently not supported (hardware limitation)

---

**auto** = 0

Automatically select the PHY based on what’s supported

**one\_mbps** = 1

1 Mbps PHY

**two\_mbps** = 2

2 Mbps PHY

```
class blatann.gap.gap_types.PeerAddress(addr_type, addr)
```

Bases: [BLEGapAddr](#)

```
class blatann.gap.gap_types.ConnectionParameters(min_conn_interval_ms, max_conn_interval_ms,
                                                  timeout_ms, slave_latency=0)
```

Bases: [BLEGapConnParams](#)

Represents the connection parameters that are sent during negotiation. This includes the preferred min/max interval range, timeout, and slave latency

```
class blatann.gap.gap_types.ActiveConnectionParameters(conn_params)
```

Bases: [object](#)

Represents the connection parameters that are currently in use with a peer device. This is similar to Connection-Parameters with the sole difference being the connection interval is not a min/max range but a single number

```
property interval_ms: float
```

**Read Only**

The connection interval, in milliseconds

```
property timeout_ms: float
```

**Read Only**

The connection timeout, in milliseconds

```
property slave_latency: int
```

**Read Only**

The slave latency (the number of connection intervals the slave is allowed to skip before being required to respond)

## **blatann.gap.generic\_access\_service module**

```
class blatann.gap.generic_access_service.GenericAccessService(ble_driver, device_name='nRF5x',
                                                             appearance=Appearance.unknown)
```

Bases: [object](#)

Class which represents the Generic Access service within the local database

```
DEVICE_NAME_MAX_LENGTH = 31
```

```
property device_name: str
```

The device name that is configured in the Generic Access service of the local GATT database

**Getter**

Gets the current device name

**Setter**

Sets the current device name. Length (after utf8 encoding) must be <= 31 bytes

```
property appearance: Appearance
```

The Appearance that is configured in the Generic Access service of the local GATT database

**Getter**

Gets the device appearance

**Setter**

Sets the device appearance

**property preferred\_peripheral\_connection\_params:** `Optional[ConnectionParameters]`

The preferred peripheral connection parameters that are configured in the Generic Access service of the local GATT Database. If not configured, returns None.

**Getter**

Gets the configured connection parameters or None if not configured

**Setter**

Sets the configured connection parameters

**update()**

**Not to be called by users**

Used internally to configure the generic access in the case that values were set before the driver was opened and configured.

**blatann.gap.scanning module**

**class** `blatann.gap.scanning.ScanParameters(interval_ms, window_ms, timeout_s, active=True)`

Bases: `BLEGapScanParams`

Class which holds scanning parameters

**validate()**

**update(window\_ms, interval\_ms, timeout\_s, active)**

**class** `blatann.gap.scanning.Scanner(ble_device)`

Bases: `object`

**property on\_scan\_received:** `Event[Scanner, ScanReport]`

Event that is raised whenever a scan report is received

**property on\_scan\_timeout:** `Event[Scanner, ScanReportCollection]`

Event that is raised when scanning completes/times out

**property is\_scanning:** `bool`

**Read Only**

Current state of scanning

**set\_default\_scan\_params(interval\_ms=200, window\_ms=150, timeout\_seconds=10, active\_scanning=True)**

Sets the default scan parameters so they do not have to be specified each time a scan is started. Reference the Bluetooth specification for valid ranges for parameters.

**Parameters**

- **interval\_ms** (`float`) – The interval which to scan for advertising packets, in milliseconds
- **window\_ms** (`float`) – How long within a single scan interval to be actively listening for advertising packets, in milliseconds
- **timeout\_seconds** (`int`) – How long to advertise for, in seconds
- **active\_scanning** (`bool`) – Whether or not to fetch scan response packets from advertisers

**start\_scan**(*scan\_parameters=None, clear\_scan\_reports=True*)

Starts a scan and returns a waitable for when the scan completes

**Parameters**

- **scan\_parameters** (*Optional[ScanParameters]*) – Optional scan parameters. Uses default if not specified
- **clear\_scan\_reports** – Flag to clear out previous scan reports

**Return type**

*ScanFinishedWaitable*

**Returns**

A Waitable which will trigger once the scan finishes based on the timeout specified. Waitable returns a ScanReportCollection of the advertising packets found

**stop()**

Stops scanning

## blatann.gap.smp module

**class** blatann.gap.smp.**SecurityLevel**(*value*)

Bases: *Enum*

Security levels used for defining GATT server characteristics

**NO\_ACCESS** = 0

**OPEN** = 1

**JUST\_WORKS** = 2

**MITM** = 3

**LESC\_MITM** = 4

**class** blatann.gap.smp.**PairingPolicy**(*value*)

Bases: *IntFlag*

An enumeration.

**allow\_all** = 0

**reject\_new\_pairing\_requests** = 1

**reject\_nonbonded\_peripheral\_requests** = 2

**reject\_bonded\_peripheral\_requests** = 4

**reject\_bonded\_device\_repairing\_requests** = 8

**reject\_peripheral\_requests** = 6

**reject\_all\_requests** = 15

**static combine**(\**policies*)

```
class blatann.gap.smp.SecurityParameters(passcode_pairing=False,  
                                         io_capabilities=BLEGapIoCaps.KEYBOARD_DISPLAY,  
                                         bond=False, out_of_band=False,  
                                         reject_pairing_requests=False, lesc_pairing=False)
```

Bases: `object`

Class representing the desired security parameters for a given connection

```
class blatann.gap.smp.SecurityManager(ble_device, peer, security_parameters)
```

Bases: `object`

Handles performing security procedures with a connected peer

**property** `on_pairing_complete`: `Event[Peer, PairingCompleteEventArgs]`

Event that is triggered when pairing completes with the peer

**Returns**

an Event which can have handlers registered to and deregistered from

**property** `on_security_level_changed`: `Event[Peer, SecurityLevelChangedEventArgs]`

Event that is triggered when the security/encryption level changes. This can be triggered from a pairing sequence or if a bonded client starts the encryption handshaking using the stored LTKs.

Note: This event is triggered before `on_pairing_complete`

**Returns**

an Event which can have handlers registered to and deregestered from

**property** `on_passkey_display_required`: `Event[Peer, PasskeyDisplayEventArgs]`

Event that is triggered when a passkey needs to be displayed to the user and depending on the pairing mode the user must confirm that keys match (`PasskeyDisplayEventArgs.match_request == True`).

---

**Note:** If multiple handlers are registered to this event, the first handler which resolves the match confirmation will set the response. All others will be ignored.

---

**Returns**

an Event which can have handlers registered to and deregistered from

**Return type**

`Event`

**property** `on_passkey_required`: `Event[Peer, PasskeyEntryEventArgs]`

Event that is triggered when a passkey needs to be entered by the user

---

**Note:** If multiple handlers are registered to this event, the first handler which resolves the passkey will set the value. All others will be ignored.

---

**Returns**

an Event which can have handlers registered to and deregistered from

**property** `on_peripheral_security_request`: `Event[Peer, PeripheralSecurityRequestEventArgs]`

Event that is triggered when the connected peripheral explicitly requests pairing/encryption to be enabled. The event provides the higher levels an opportunity to accept, reject, or force re-pair with the peripheral.

If no handler is registered to this event, pairing requests will be accepted unless the `reject_pairing_requests` parameter is set.

---

**Note:** If a handler is registered to this event, it **must** respond with one of the options (accept/reject/repair).

---

---

**Note:** If multiple handlers are registered to this event, the first handler to respond is the response used. All other inputs will be ignored

---

### Returns

Event that is triggered when the peripheral requests a secure connection

**property on\_pairing\_request\_rejected:** *Event[Peer, PairingRejectedEventArgs]*

Event that's emitted when a pairing request is rejected locally, either due to the user event handler or due to the rejection policy set in the security parameters

### Returns

Event that is triggered when a pairing request is rejected

**property is\_previously\_bonded:** *bool*

Gets if the peer this security manager is for was bonded in a previous connection

### Returns

True if previously bonded, False if not

**property pairing\_in\_process:** *bool*

Gets whether or not pairing/encryption is currently in process

**property security\_level:** *SecurityLevel*

Gets the current security level of the connection

**property security\_params:** *SecurityParameters*

Gets the security parameters structure

**set\_security\_params**(*passcode\_pairing, io\_capabilities, bond, out\_of\_band, reject\_pairing\_requests=False, lesc\_pairing=False*)

Sets the security parameters to use with the peer

### Parameters

- **passcode\_pairing** (*bool*) – Flag indicating that passcode pairing is required
- **io\_capabilities** (*BLEGapIoCaps*) – The input/output capabilities of this device
- **bond** (*bool*) – Flag indicating that long-term bonding should be performed
- **out\_of\_band** (*bool*) – Flag indicating if out-of-band pairing is supported
- **reject\_pairing\_requests** (*Union[bool, PairingPolicy]*) – Flag indicating that all security requests by the peer should be rejected
- **lesc\_pairing** (*bool*) – Flag indicating that LE Secure Pairing methods are supported

**pair**(*force\_repairing=False*)

Starts the pairing process with the peer with the set security parameters.

If the peer is already bonded, initiates the encryption process unless `force_repairing` is set to True

If the peer is a central and we are a local device, sends the peripheral security request to the central so they can start the pairing/encryption process

**Return type**

`EventWaitable[Peer, PairingCompleteEventArgs]`

**Returns**

A waitable that will trigger when pairing is complete

**use\_debug\_lesc\_key()**

Changes the security settings to use the debug public/private key-pair for future LESC pairing interactions. The key is defined in the Core Bluetooth Specification v4.2 Vol.3, Part H, Section 2.3.5.6.

**Warning:** Using this key allows Bluetooth sniffers to be able to decode the encrypted traffic over the air

**delete\_bonding\_data()**

Deletes the bonding data for the peer, if any. Cannot be called during pairing, will throw an `InvalidOperationException`

## blatann.gap.smp\_crypto module

`blatann.gap.smp_crypto.lesc_pubkey_to_raw(public_key, little_endian=True)`

Converts from a python public key to the raw (x, y) bytes for the nordic

**Return type**

`bytearray`

`blatann.gap.smp_crypto.lesc_privkey_to_raw(private_key, little_endian=True)`

**Return type**

`bytearray`

`blatann.gap.smp_crypto.lesc_pubkey_from_raw(raw_key, little_endian=True)`

Converts from raw (x, y) bytes to a public key that can be used for the DH request

**Return type**

`EllipticCurvePublicKey`

`blatann.gap.smp_crypto.lesc_privkey_from_raw(raw_priv_key, raw_pub_key, little_endian=True)`

**Return type**

`EllipticCurvePrivateKey`

`blatann.gap.smp_crypto.lesc_generate_private_key()`

Generates a new private key that can be used for LESC pairing

**Return type**

`EllipticCurvePrivateKey`

**Returns**

The generated private key

`blatann.gap.smp_crypto.lesc_compute_dh_key(private_key, peer_public_key, little_endian=False)`

Computes the DH key for LESC pairing given our private key and the peer's public key

**Parameters**

- **private\_key** (`EllipticCurvePrivateKey`) – Our private key
- **peer\_public\_key** (`EllipticCurvePublicKey`) – The peer’s public key
- **little\_endian** – whether or not to return the shared secret in little endian

**Return type**`bytes`**Returns**

The shared secret

`blatann.gap.smp_crypto.ble_ah(key, p_rand)`

Function for calculating the ah() hash function described in Bluetooth core specification 4.2 section 3.H.2.2.2.

This is used for resolving private addresses where a private address is `prand[3] || aes-128(irk, prand[3]) % 2^24`**Parameters**

- **key** (`bytes`) – the IRK to use, in big endian format
- **p\_rand** (`bytes`) – The random component, first 3 bytes of the address

**Return type**`bytes`**Returns**

The last 3 bytes of the encrypted hash

`blatann.gap.smp_crypto.private_address_resolves(peer_addr, irk)`

Checks if the given peer address can be resolved with the IRK

Private Resolvable Peer Addresses are in the format `[4x:xx:xx:yy:yy:yy]`, where `4x:xx:xx` is a random number hashed with the IRK to generate `yy:yy:yy`. This function checks if the random number portion hashed with the IRK equals the hashed part of the address**Parameters**

- **peer\_addr** (`PeerAddress`) – The peer address to check
- **irk** (`bytes`) – The identity resolve key to try

**Return type**`bool`**Returns**

True if it resolves, False if not

**blatann.gatt package**`blatann.gatt.logger = <Logger blatann.gatt (INFO)>`

The default MTU size that’s used when a connection is established

`blatann.gatt.MTU_SIZE_DEFAULT = 23`

The minimum allowed MTU size

`blatann.gatt.MTU_SIZE_MINIMUM = 23`

The ideal MTU size to use when using the maximum link-layer Data Length Extension setting (251)

`blatann.gatt.DLE_OVERHEAD = 4`

Status codes that can be returned during GATT Operations (reads, writes, etc.)



`blatann.gatt.GattStatusCode`

The two notification types (notification, indication) used when a characteristic is notified from a peripheral

**class** `blatann.gatt.ServiceType(value)`

Bases: `IntEnum`

An enumeration.

**PRIMARY** = 1

**SECONDARY** = 2

**class** `blatann.gatt.SubscriptionState(value)`

Bases: `IntEnum`

Defines the different subscription states/types for a characteristic

**NOT\_SUBSCRIBED** = 0

**NOTIFY** = 1

**INDICATION** = 2

**classmethod** `to_buffer(value)`

Converts to a little-endian uint16 buffer to be written over BLE

**classmethod** `from_buffer(buf)`

Converts from a little-endian uint16 buffer received over BLE to the subscription state

**class** `blatann.gatt.CharacteristicProperties(read=True, write=False, notify=False, indicate=False, broadcast=False, write_no_response=False, signed_write=False)`

Bases: `object`

**class** `blatann.gatt.Attribute(uuid, handle, value=b'', string_encoding='utf8')`

Bases: `object`

Represents a single attribute which lives inside a Characteristic (both remote and local)

**property** `uuid: Uuid`

The attribute's UUID

**property** `handle: int`

The attribute's handle

**property** `value: bytes`

Gets the current value of the attribute

**property** `string_encoding: str`

The default method for encoding strings into bytes when a string is provided as a value

**class** `blatann.gatt.Characteristic(ble_device, peer, uuid, properties, attributes=None, default_string_encoding='utf8')`

Bases: `object`

Abstract class that represents a BLE characteristic (both remote and local).

**class** `blatann.gatt.Service(ble_device, peer, uuid, service_type, start_handle=0, end_handle=0)`

Bases: `object`

Abstract class that represents a BLE Service (both remote and local)

```
class blatann.gatt.GattDatabase(ble_device, peer)
```

Bases: `object`

Abstract class that represents a BLE Database (both remote and local)

```
class blatann.gatt.PresentationFormat(fmt, exponent, unit, namespace=0, description=0)
```

Bases: `BleCompoundDataType`

```
data_stream_types = [<class 'blatann.services.ble_data_types.Uint8'>, <class  
'blatann.services.ble_data_types.Int8'>, <class  
'blatann.services.ble_data_types.Uint16'>, <class  
'blatann.services.ble_data_types.Uint8'>, <class  
'blatann.services.ble_data_types.Uint16'>]
```

```
encode()
```

Return type

`BleDataStream`

```
classmethod decode(stream)
```

Returns

The values decoded from the stream

Return type

`tuple`

```
static try_get_enum(value, enum_type)
```

## Submodules

### blatann.gatt.gattc module

```
class blatann.gatt.gattc.GattcCharacteristic(ble_device, peer, uuid, properties, decl_attr, value_attr,  
                                              cccd_attr=None, attributes=None)
```

Bases: `Characteristic`

Represents a characteristic that lives within a service in the server's GATT database.

This class is normally not instantiated directly and instead created when the database is discovered via `Peer.discover_services()`

property declaration\_attribute: `GattcAttribute`

Read Only

Gets the declaration attribute of the characteristic

property value\_attribute: `GattcAttribute`

Read Only

Gets the value attribute of the characteristic

property value: `bytes`

Read Only

The current value of the characteristic. This is updated through read, write, and notify operations

**property readable:** `bool`

Read Only

Gets if the characteristic can be read from

**property writable:** `bool`

Read Only

Gets if the characteristic can be written to

**property writable\_without\_response:** `bool`

Read Only

Gets if the characteristic accepts write commands that don't require a confirmation response

**property subscribable:** `bool`

Read Only

Gets if the characteristic can be subscribed to

**property subscribable\_indications:** `bool`

Read Only

Gets if the characteristic can be subscribed to using indications

**property subscribable\_notifications:** `bool`

Read Only

Gets if the characteristic can be subscribed to using notifications

**property subscribed:** `bool`

Read Only

Gets if the characteristic is currently subscribed to

**property attributes:** `Iterable[GattcAttribute]`

Read Only

Returns the list of all attributes/descriptors that reside in the characteristic. This includes the declaration attribute, value attribute, and descriptors (CCCD, Name, etc.)

**property string\_encoding:** `str`

The default method for encoding strings into bytes when a string is provided as a value

**Getter**

Gets the current string encoding for the characteristic

**Setter**

Sets the string encoding for the characteristic

**property on\_read\_complete:** `Event[GattcCharacteristic, ReadCompleteEventArgs]`

Event that is raised when a read operation from the characteristic is completed

**property on\_write\_complete:** `Event[GattcCharacteristic, WriteCompleteEventArgs]`

Event that is raised when a write operation to the characteristic is completed

**property on\_notification\_received:** `Event[GattcCharacteristic, NotificationReceivedEventArgs]`

Event that is raised when an indication or notification is received on the characteristic

**subscribe**(*on\_notification\_handler*, *prefer\_indications=False*)

Subscribes to the characteristic's indications or notifications, depending on what's available and the *prefer\_indications* setting. Returns a Waitable that triggers when the subscription on the peripheral finishes.

**Parameters**

- **on\_notification\_handler** (Callable[[*GattcCharacteristic*, *NotificationReceivedEventArgs*], None]) – The handler to be called when an indication or notification is received from the peripheral. Must take two parameters: (*GattcCharacteristic* this, *NotificationReceivedEventArgs* event args)
- **prefer\_indications** – If the peripheral supports both indications and notifications, will subscribe to indications instead of notifications

**Return type**

*EventWaitable*[*GattcCharacteristic*, *SubscriptionWriteCompleteEventArgs*]

**Returns**

A Waitable that will trigger when the subscription finishes

**Raises**

*InvalidOperationException* if the characteristic cannot be subscribed to (characteristic does not support indications or notifications)

**unsubscribe**()

Unsubscribes from indications and notifications from the characteristic and clears out all handlers for the characteristic's *on\_notification* event handler. Returns a Waitable that triggers when the unsubscription finishes.

**Return type**

*EventWaitable*[*GattcCharacteristic*, *SubscriptionWriteCompleteEventArgs*]

**Returns**

A Waitable that will trigger when the unsubscription operation finishes

**Raises**

*InvalidOperationException* if characteristic cannot be subscribed to (characteristic does not support indications or notifications)

**read**()

Initiates a read of the characteristic and returns a Waitable that triggers when the read finishes with the data read.

**Return type**

*EventWaitable*[*GattcCharacteristic*, *ReadCompleteEventArgs*]

**Returns**

A waitable that will trigger when the read finishes

**Raises**

*InvalidOperationException* if characteristic not readable

**write**(*data*)

Performs a write request of the data provided to the characteristic and returns a Waitable that triggers when the write completes and the confirmation response is received from the other device.

**Parameters**

**data** (*str* or *bytes* or *bytearray*) – The data to write. Can be a string, bytes, or anything that can be converted to bytes

**Return type**

*EventWaitable*[*GattcCharacteristic*, *WriteCompleteEventArgs*]

**Returns**

A waitable that returns when the write finishes

**Raises**

InvalidOperationException if characteristic is not writable

**write\_without\_response**(*data*)

Performs a write command, which does not require the peripheral to send a confirmation response packet. This is a faster but lossy operation in the case that the packet is dropped/never received by the peer. This returns a waitable that triggers when the write is transmitted to the peripheral device.

---

**Note:** Data sent without responses must fit within a single MTU minus 3 bytes for the operation overhead.

---

**Parameters**

**data** (*str* or *bytes* or *bytearray*) – The data to write. Can be a string, bytes, or anything that can be converted to bytes

**Return type**

*EventWaitable[GattcCharacteristic, WriteCompleteEventArgs]*

**Returns**

A waitable that returns when the write finishes

**Raises**

InvalidOperationException if characteristic is not writable without responses

**find\_descriptor**(*uuid*)

Searches for the descriptor/attribute matching the UUID provided and returns the attribute. If not found, returns None. If multiple attributes with the same UUID exist in the characteristic, this returns the first attribute found.

**Parameters**

**uuid** (*Uuid*) – The UUID to search for

**Return type**

*Optional[GattcAttribute]*

**Returns**

The descriptor attribute, if found

**class** `blatann.gatt.gattc.GattcService`(*ble\_device*, *peer*, *uuid*, *service\_type*, *start\_handle=0*, *end\_handle=0*)

Bases: *Service*

Represents a service that lives within the server's GATT database.

This class is normally not instantiated directly and instead created when the database is discovered via *Peer.discover\_services()*

**property characteristics:** *List[GattcCharacteristic]*

Gets the list of characteristics within the service

**find\_characteristic**(*characteristic\_uuid*)

Finds the characteristic matching the given UUID inside the service. If not found, returns None. If multiple characteristics with the same UUID exist within the service, this will return the first one found.

**Parameters**

**characteristic\_uuid** (*Uuid*) – The UUID of the characteristic to find

**Return type**`Optional[GattcCharacteristic]`**Returns**

The characteristic if found, otherwise None

**class** `blatann.gatt.gattc.GattcDatabase`(*ble\_device, peer, write\_no\_resp\_queue\_size=1*)Bases: `GattDatabase`

Represents a remote GATT Database which lives on a connected peripheral. Contains all discovered services, characteristics, and descriptors

**property services:** `List[GattcService]`

Gets the list of services within the database

**find\_service**(*service\_uuid*)

Finds the service matching the given UUID inside the database. If not found, returns None. If multiple services with the same UUID exist in the database, this will return the first service found.

**Parameters****service\_uuid** (`Uuid`) – The UUID of the service to find**Return type**`Optional[GattcService]`**Returns**

The service if found, otherwise None

**find\_characteristic**(*characteristic\_uuid*)

Finds the characteristic matching the given UUID inside the database. If not found, returns None. If multiple characteristics with the same UUID exist in the database, this will return the first characteristic found.

**Parameters****characteristic\_uuid** (`blatann.uuid.Uuid`) – The UUID of the characteristic to find**Returns**

The characteristic if found, otherwise None

**Return type**`GattcCharacteristic`**iter\_characteristics**()

Iterates through all the characteristics in the database

**Return type**`Iterable[GattcCharacteristic]`**Returns**

An iterable of the characteristics in the database

**blatann.gatt.gattc\_attribute module**

```
class blatann.gatt.gattc_attribute.GattcAttribute(uuid, handle, read_write_manager,
                                              initial_value=b'', string_encoding='utf8')
```

Bases: *Attribute*

Represents a client-side interface to a single attribute which lives inside a Characteristic

**property on\_read\_complete:** *Event[GattcAttribute, ReadCompleteEventArgs]*

Event that is triggered when a read from the attribute is completed

**property on\_write\_complete:** *Event[GattcAttribute, WriteCompleteEventArgs]*

Event that is triggered when a write to the attribute is completed

**read()**

Performs a read of the attribute and returns a Waitable that executes when the read finishes with the data read.

**Return type**

*IdBasedEventWaitable[GattcAttribute, ReadCompleteEventArgs]*

**Returns**

A waitable that will trigger when the read finishes

**write(data, with\_response=True)**

Initiates a write of the data provided to the attribute and returns a Waitable that executes when the write completes and the confirmation response is received from the other device.

**Parameters**

- **data** (*str* or *bytes* or *bytearray*) – The data to write. Can be a string, bytes, or anything that can be converted to bytes
- **with\_response** – Used internally for characteristics that support write without responses. Should always be true for any other case (descriptors, etc.).

**Return type**

*IdBasedEventWaitable[GattcAttribute, WriteCompleteEventArgs]*

**Returns**

A waitable that returns when the write finishes

**update(value)**

Used internally to update the value after data is received from another means, i.e. Indication/notification. Should not be called by the user.

**blatann.gatt.gatts module**

```
class blatann.gatt.gatts.GattsUserDescriptionProperties(value, write=False,
                                                         security_level=SecurityLevel.OPEN,
                                                         max_length=0, variable_length=False)
```

Bases: *GattsAttributeProperties*

Properties used to configure the User Description characteristic descriptor.

The most basic, set-once, read-only usage of this is `GattsUserDescriptionProperties("my description")`

```
class blatann.gatt.gatts.GattsCharacteristicProperties(read=True, write=False, notify=False,
                                                    indicate=False, broadcast=False,
                                                    write_no_response=False,
                                                    signed_write=False,
                                                    security_level=SecurityLevel.OPEN,
                                                    max_length=20, variable_length=True,
                                                    sccd=False, user_description=None,
                                                    presentation_format=None,
                                                    cccd_write_security_level=SecurityLevel.OPEN)
```

Bases: [CharacteristicProperties](#)

Properties for Gatt Server characteristics

```
class blatann.gatt.gatts.GattsCharacteristic(ble_device, peer, uuid, properties, value_handle,
                                           cccd_handle, sccd_handle, user_desc_handle,
                                           notification_manager, value=b'', prefer_indications=True,
                                           string_encoding='utf8')
```

Bases: [Characteristic](#)

Represents a single characteristic within a service. This class is usually not instantiated directly; it is added to a service through [GattService.add\\_characteristic\(\)](#)

**set\_value**(value, notify\_client=False)

Sets the value of the characteristic.

#### Parameters

- **value** – The value to set to. Must be an iterable type such as a str, bytes, or list of uint8 values, or a [BleDataStream](#) object. Length must be less than or equal to the characteristic's max length. If a string is given, it will be encoded using the `string_encoding` property of the characteristic.
- **notify\_client** – Flag whether or not to notify the client. If indications and notifications are not set up for the characteristic, will raise an [InvalidOperationException](#)

#### Raises

[InvalidOperationException](#) if value length is too long, or notify client set and characteristic is not notifiable

#### Raises

[InvalidStateException](#) if the client is not currently subscribed to the characteristic

#### Return type

[Optional](#)[[IdBasedEventWaitable](#)[[GattsCharacteristic](#),  
[NotificationCompleteEventArgs](#)]]

#### Returns

If `notify_client` is true, this method will return the waitable for when the notification is sent to the client

**notify**(data)

Notifies the client with the data provided without setting the data into the characteristic value. If data is not provided (None), will notify with the currently-set value of the characteristic

#### Parameters

**data** – Optional data to notify the client with. If supplied, must be an iterable type such as a str, bytes, or list of uint8 values, or a [BleDataStream](#) object. Length must be less than or equal to the characteristic's max length. If a string is given, it will be encoded using the `string_encoding` property of the characteristic.



**Raises**

`InvalidStateException` if the client is not subscribed to the characteristic

**Raises**

`InvalidOperationException` if the characteristic is not configured for notifications/indications

**Return type**

`IdBasedEventWaitable[GattsCharacteristic, NotificationCompleteEventArgs]`

**Returns**

An `EventWaitable` that will trigger when the notification is successfully sent to the client. The waitable also contains the ID of the sent notification which is used in the `on_notify_complete` event

**add\_descriptor**(*uuid*, *properties*, *initial\_value=b''*, *string\_encoding='utf8'*)

Creates and adds a descriptor to the characteristic

---

**Note:** Due to limitations of the BLE stack, the CCCD, SCCD, User Description, Extended Properties, and Presentation Format descriptors cannot be added through this method. They must be added through the `GattsCharacteristicProperties` fields when creating the characteristic.

---

**Parameters**

- **uuid** (*Uuid*) – The UUID of the descriptor to add, and cannot be the UUIDs of any of the reserved descriptor UUIDs in the note
- **properties** (*GattsAttributeProperties*) – The properties of the descriptor
- **initial\_value** – The initial value to set the descriptor to
- **string\_encoding** – The string encoding to use, if a string is set

**Return type**

*GattsAttribute*

**Returns**

the descriptor that was created and added to the characteristic

**add\_constant\_value\_descriptor**(*uuid*, *value*, *security\_level=SecurityLevel.OPEN*)

Adds a descriptor to the characteristic which is a constant, read-only value that cannot be updated after this call. This is a simplified parameter set built on top of `add_descriptor()` for this common use-case.

---

**Note:** See note on `add_descriptor()` for limitations on descriptors that can be added through this method.

---

**Parameters**

- **uuid** (*Uuid*) – The UUID of the descriptor to add
- **value** (*bytes*) – The value to set the descriptor to
- **security\_level** – The security level for the descriptor

**Return type**

*GattsAttribute*

**Returns**

The descriptor that was created and added to the characteristic

**property max\_length:** `int`

**Read Only**

The max possible the value the characteristic can be set to

**property notifiable:** `bool`

**Read Only**

Gets if the characteristic is set up to asynchronously notify clients via notifications or indications

**property value:** `bytes`

**Read Only**

Gets the current value of the characteristic. Value is updated using `set_value()`

**property client\_subscribed:** `bool`

**Read Only**

Gets if the client is currently subscribed (notify or indicate) to this characteristic

**property attributes:** `Iterable[GattsAttribute]`

**Read Only**

Gets all of the attributes and descriptors associated with this characteristic

**property user\_description:** `Optional[GattsAttribute]`

**Read Only**

Gets the User Description attribute for the characteristic if set in the properties. If the user description was not configured for the characteristic, returns None

**property sccd:** `Optional[GattsAttribute]`

**Read Only**

Gets the Server Characteristic Configuration Descriptor (SCCD) attribute if set in the properties. If the SCCD was not configured for the characteristic, returns None

**property presentation\_format:** `Optional[PresentationFormat]`

**Read Only**

Gets the presentation format that was set for the characteristic. If the presentation format was not configured for the characteristic, returns None

**property string\_encoding:** `str`

The default method for encoding strings into bytes when a string is provided as a value

**Getter**

Gets the string encoding in use

**Setter**

Sets the string encoding to use

**property on\_write:** `Event[GattsCharacteristic, WriteEventArgs]`

Event generated whenever a client writes to this characteristic.

**Returns**

an Event which can have handlers registered to and deregistered from

**property on\_read:** [Event](#)[[GattsCharacteristic](#), [None](#)]

Event generated whenever a client requests to read from this characteristic. At this point, the application may choose to update the value of the characteristic to a new value using `set_value`.

A good example of this is a “system time” characteristic which reports the applications system time in seconds. Instead of updating this characteristic every second, it can be “lazily” updated only when read from.

NOTE: if there are multiple handlers subscribed to this and each set the value differently, it may cause undefined behavior.

#### Returns

an Event which can have handlers registered to and deregistered from

**property on\_subscription\_change:** [Event](#)[[GattsCharacteristic](#), [SubscriptionStateChangeEventArgs](#)]

Event that is generated whenever a client changes its subscription state of the characteristic (notify, indicate, none).

#### Returns

an Event which can have handlers registered to and deregistered from

**property on\_notify\_complete:** [Event](#)[[GattsCharacteristic](#), [NotificationCompleteEventArgs](#)]

Event that is generated when a notification or indication sent to the client successfully

#### Returns

an event which can have handlers registered to and deregistered from

**class** `blatann.gatt.gatts.GattsService`(*ble\_device*, *peer*, *uuid*, *service\_type*, *notification\_manager*, *start\_handle=0*, *end\_handle=0*)

Bases: [Service](#)

Represents a registered GATT service that lives locally on the device.

This class is usually not instantiated directly and is instead created through [GattsDatabase.add\\_service\(\)](#).

**property characteristics:** [List](#)[[GattsCharacteristic](#)]

#### Read Only

Gets the list of characteristics in this service.

Characteristics are added through [add\\_characteristic\(\)](#)

**add\_characteristic**(*uuid*, *properties*, *initial\_value=b''*, *prefer\_indications=True*, *string\_encoding='utf8'*)

Adds a new characteristic to the service

#### Parameters

- **uuid** ([Uuid](#)) – The UUID of the characteristic to add
- **properties** ([GattsCharacteristicProperties](#)) – The characteristic’s properties
- **initial\_value** (*str* or *list* or *bytearray*) – The initial value of the characteristic. May be a string, bytearray, or list of ints
- **prefer\_indications** – Flag for choosing indication/notification if a characteristic has both indications and notifications available
- **string\_encoding** – The encoding method to use when a string value is provided (utf8, ascii, etc.)

**Returns**

The characteristic just added to the service

**Return type**

*GattsCharacteristic*

```
class blatann.gatt.gatts.GattsDatabase(ble_device, peer, notification_hardware_queue_size=1)
```

Bases: *GattDatabase*

Represents the entire GATT server that lives locally on the device which clients read from and write to

**property services:** *List*[*GattsService*]

**Read Only**

The list of services registered in the database

**iter\_services()**

Iterates through all of the registered services in the database

**Return type**

*Iterable*[*GattsService*]

**Returns**

Generator of the database's services

```
add_service(uuid, service_type=ServiceType.PRIMARY)
```

Adds a service to the local database

**Parameters**

- **uuid** (*Uuid*) – The UUID for the service
- **service\_type** – The type of service (primary or secondary)

**Return type**

*GattsService*

**Returns**

The added and newly created service

```
clear_pending_notifications()
```

Clears all pending notifications that are queued to be sent to the client

**blatann.gatt.gatts\_attribute module**

```
class blatann.gatt.gatts_attribute.GattsAttributeProperties(read=True, write=False,
                                                           security_level=SecurityLevel.OPEN,
                                                           max_length=20,
                                                           variable_length=True,
                                                           read_auth=False, write_auth=False)
```

Bases: *object*

```
class blatann.gatt.gatts_attribute.GattsAttribute(ble_device, peer, parent, uuid, handle, properties,
                                                  initial_value=b'', string_encoding='utf8')
```

Bases: *Attribute*

Represents the server-side interface of a single attribute which lives inside a Characteristic.

**property parent:** *GattsCharacteristic*

**Read Only**

Gets the parent characteristic which owns this attribute

**property max\_length:** *int*

**Read Only**

The max possible length data the attribute can be set to

**property read\_in\_process:** *bool*

**Read Only**

Gets whether or not the client is in the process of reading out this attribute

**set\_value**(*value*)

Sets the value of the attribute.

**Parameters**

**value** – The value to set to. Must be an iterable type such as a str, bytes, or list of uint8 values, or a *BleDataStream* object. Length must be less than the attribute’s max length. If a str is given, it will be encoded using the *string\_encoding* property.

**Raises**

*InvalidOperationException* if value length is too long

**get\_value**()

Fetches the attribute’s value from hardware and updates the local copy. This isn’t often necessary and should instead use the value property to avoid unnecessary reads from the hardware.

**Return type**

*bytes*

**property on\_write:** *Event[GattsAttribute, WriteEventArgs]*

Event generated whenever a client writes to this attribute.

**Returns**

an Event which can have handlers registered to and deregistered from

**property on\_read:** *Event[GattsAttribute, None]*

Event generated whenever a client requests to read from this attribute. At this point, the application may choose to update the value of the attribute to a new value using *set\_value*.

---

**Note:** This will only be triggered if the attribute was configured with the *read\_auth* property

---

A good example of using this is a “system time” characteristic which reports the application’s current system time in seconds. Instead of updating this characteristic every second, it can be “lazily” updated only when read.

NOTE: if there are multiple handlers subscribed to this and each set the value differently, it may cause undefined behavior.

**Returns**

an Event which can have handlers registered to and deregistered from

### blatann.gatt.managers module

```
class blatann.gatt.managers.GattcOperationManager(ble_device, peer, reader, writer,
                                                    write_no_response_queue_size=1)
```

Bases: `object`

**read**(handle, callback)

**write**(handle, value, callback, with\_response=True)

**clear\_all**()

```
class blatann.gatt.managers.GattsOperationManager(ble_device, peer, notification_queue_size=1)
```

Bases: `object`

**notify**(characteristic, handle, event\_on\_complete, data=None)

**clear\_all**()

### blatann.gatt.reader module

```
class blatann.gatt.reader.GattcReadCompleteEventArgs(handle, status, data)
```

Bases: `EventArgs`

```
class blatann.gatt.reader.GattcReader(ble_device, peer)
```

Bases: `object`

Class which implements the state machine for completely reading a peripheral's attribute

#### **property on\_read\_complete**

Event that is emitted when a read completes on an attribute handle.

Handler args: (int attribute\_handle, gatt.GattStatusCode, bytes data\_read)

#### **Returns**

an Event which can have handlers registered to and deregistered from

#### **Return type**

`Event`

**read**(handle)

Reads the attribute value from the handle provided. Can only read from a single attribute at a time. If a read is in progress, raises an `InvalidStateException`

#### **Parameters**

**handle** – the attribute handle to read

#### **Returns**

A waitable that will fire when the read finishes. See `on_read_complete` for the values returned from the waitable

#### **Return type**

`EventWaitable`

**blatann.gatt.service\_discovery module**

**class** blatann.gatt.service\_discovery.DatabaseDiscoverer(*ble\_device, peer*)

Bases: `object`

**property** on\_discovery\_complete

**Return type**

*Event[blatann.peer.Peripheral, DatabaseDiscoveryCompleteEventArgs]*

**start()**

**blatann.gatt.writer module**

**class** blatann.gatt.writer.GattcWriteCompleteEventArgs(*handle, status, data*)

Bases: *EventArgs*

**class** blatann.gatt.writer.GattcWriter(*ble\_device, peer*)

Bases: `object`

Class which implements the state machine for writing a value to a peripheral's attribute

**property** on\_write\_complete

Event that is emitted when a write completes on an attribute handler

Handler args: (int attribute\_handle, gatt.GattStatusCode, bytearray data\_written)

**Returns**

an Event which can have handlers registered to and deregistered from

**Return type**

*Event*

**write**(*handle, data*)

Writes data to the attribute at the handle provided. Can only write to a single attribute at a time. If a write is in progress, raises an InvalidStateException

**Parameters**

- **handle** – The attribute handle to write
- **data** – The data to write

**Returns**

A Waitable that will fire when the write finishes. see on\_write\_complete for the values returned from the waitable

**Return type**

*EventWaitable*

## blatann.nrf package

### Subpackages

#### blatann.nrf.nrf\_events package

`blatann.nrf.nrf_events.event_decode(ble_event)`

### Submodules

#### blatann.nrf.nrf\_events.gap\_events module

**class** `blatann.nrf.nrf_events.gap_events.GapEvt(conn_handle)`

Bases: [\*BLEEvent\*](#)

**class** `blatann.nrf.nrf_events.gap_events.GapEvtRssiChanged(conn_handle, rssi)`

Bases: [\*GapEvt\*](#)

**evt\_id** = 28

**classmethod** `from_c(event)`

**class** `blatann.nrf.nrf_events.gap_events.GapEvtAdvReport(conn_handle, peer_addr, rssi, adv_type, adv_data)`

Bases: [\*GapEvt\*](#)

**evt\_id** = 29

**get\_device\_name()**

**classmethod** `from_c(event)`

**class** `blatann.nrf.nrf_events.gap_events.GapEvtTimeout(conn_handle, src)`

Bases: [\*GapEvt\*](#)

**evt\_id** = 27

**classmethod** `from_c(event)`

**class** `blatann.nrf.nrf_events.gap_events.GapEvtConnParamUpdateRequest(conn_handle, conn_params)`

Bases: [\*GapEvt\*](#)

**evt\_id** = 31

**classmethod** `from_c(event)`

**class** `blatann.nrf.nrf_events.gap_events.GapEvtConnParamUpdate(conn_handle, conn_params)`

Bases: [\*GapEvt\*](#)

**evt\_id** = 18

**classmethod** `from_c(event)`



```
class blatann.nrf.nrf_events.gap_events.GapEvtConnected(conn_handle, peer_addr, role,  
                                                         conn_params)  
  
    Bases: GapEvt  
    evt_id = 16  
  
    classmethod from_c(event)  
  
class blatann.nrf.nrf_events.gap_events.GapEvtDisconnected(conn_handle, reason)  
    Bases: GapEvt  
    evt_id = 17  
  
    classmethod from_c(event)  
  
class blatann.nrf.nrf_events.gap_events.GapEvtDataLengthUpdate(conn_handle, max_tx_octets,  
                                                                max_rx_octets, max_tx_time_us,  
                                                                max_rx_time_us)  
  
    Bases: GapEvt  
    evt_id = 36  
  
    classmethod from_c(event)  
  
class blatann.nrf.nrf_events.gap_events.GapEvtDataLengthUpdateRequest(conn_handle,  
                                                                        max_tx_octets,  
                                                                        max_rx_octets,  
                                                                        max_tx_time_us,  
                                                                        max_rx_time_us)  
  
    Bases: GapEvt  
    evt_id = 35  
  
    classmethod from_c(event)  
  
class blatann.nrf.nrf_events.gap_events.GapEvtPhyUpdate(conn_handle, status, tx_phy, rx_phy)  
    Bases: GapEvt  
    evt_id = 34  
  
    classmethod from_c(event)  
  
class blatann.nrf.nrf_events.gap_events.GapEvtPhyUpdateRequest(conn_handle, tx_phy, rx_phy)  
    Bases: GapEvt  
    evt_id = 33  
  
    classmethod from_c(event)
```

**blatann.nrf.nrf\_events.gatt\_events module**

```
class blatann.nrf.nrf_events.gatt_events.GattEvt(conn_handle)
```

Bases: *BLEEvent*

```
class blatann.nrf.nrf_events.gatt_events.GattcEvt(conn_handle)
```

Bases: *GattEvt*

```
class blatann.nrf.nrf_events.gatt_events.GattsEvt(conn_handle)
```

Bases: *GattEvt*

```
class blatann.nrf.nrf_events.gatt_events.GattcEvtReadResponse(conn_handle, status, error_handle,
                                                                attr_handle, offset, data)
```

Bases: *GattcEvt*

**evt\_id** = 54

**classmethod** **from\_c**(event)

```
class blatann.nrf.nrf_events.gatt_events.GattcEvtHvx(conn_handle, status, error_handle,
                                                       attr_handle, hvx_type, data)
```

Bases: *GattcEvt*

**evt\_id** = 57

**classmethod** **from\_c**(event)

```
class blatann.nrf.nrf_events.gatt_events.GattcEvtWriteCmdTxComplete(conn_handle, count)
```

Bases: *GattcEvt*

**evt\_id** = 60

**classmethod** **from\_c**(event)

```
class blatann.nrf.nrf_events.gatt_events.GattcEvtWriteResponse(conn_handle, status,
                                                                error_handle, attr_handle,
                                                                write_op, offset, data)
```

Bases: *GattcEvt*

**evt\_id** = 56

**classmethod** **from\_c**(event)

```
class blatann.nrf.nrf_events.gatt_events.GattcEvtPrimaryServiceDiscoveryResponse(conn_handle,
                                                                                    status,
                                                                                    services)
```

Bases: *GattcEvt*

**evt\_id** = 48

**classmethod** **from\_c**(event)

```
class blatann.nrf.nrf_events.gatt_events.GattcEvtCharacteristicDiscoveryResponse(conn_handle,
                                                                                    status,
                                                                                    character-
                                                                                    istics)
```

Bases: *GattcEvt*

```

    evt_id = 50

    classmethod from_c(event)

class blatann.nrf.nrf_events.gatt_events.GattcEvtDescriptorDiscoveryResponse(conn_handle,
                                                                              status,
                                                                              descriptions)

    Bases: GattcEvt

    evt_id = 51

    classmethod from_c(event)

class blatann.nrf.nrf_events.gatt_events.GattcEvtAttrInfoDiscoveryResponse(conn_handle,
                                                                              status,
                                                                              attr_info16=None,
                                                                              attr_info128=None)

    Bases: GattcEvt

    evt_id = 52

    classmethod from_c(event)

class blatann.nrf.nrf_events.gatt_events.GattcEvtMtuExchangeResponse(conn_handle, server_mtu)

    Bases: GattcEvt

    evt_id = 58

    classmethod from_c(event)

class blatann.nrf.nrf_events.gatt_events.GattcEvtTimeout(conn_handle, source)

    Bases: GattcEvt

    evt_id = 59

    classmethod from_c(event)

class blatann.nrf.nrf_events.gatt_events.GattsEvtSysAttrMissing(conn_handle, hint)

    Bases: GattsEvt

    evt_id = 82

    classmethod from_c(event)

class blatann.nrf.nrf_events.gatt_events.GattsEvtWrite(conn_handle, attr_handle, uuid,
                                                         write_operand, auth_required, offset, data)

    Bases: GattsEvt

    evt_id = 80

    classmethod from_c(event)

    classmethod from_auth_request(conn_handle, write_event)

class blatann.nrf.nrf_events.gatt_events.GattsEvtRead(conn_handle, attr_handle, uuid, offset)

    Bases: GattsEvt

    classmethod from_auth_request(conn_handle, read_event)

```

```
class blatann.nrf.nrf_events.gatt_events.GattsEvtReadWriteAuthorizeRequest(conn_handle,
                                                                           read=None,
                                                                           write=None)
```

Bases: *GattsEvt*

evt\_id = 81

classmethod from\_c(event)

```
class blatann.nrf.nrf_events.gatt_events.GattsEvtHandleValueConfirm(conn_handle, attr_handle)
```

Bases: *GattsEvt*

evt\_id = 83

classmethod from\_c(event)

```
class blatann.nrf.nrf_events.gatt_events.GattsEvtNotificationTxComplete(conn_handle,
                                                                           tx_count)
```

Bases: *GattsEvt*

evt\_id = 87

classmethod from\_c(event)

```
class blatann.nrf.nrf_events.gatt_events.GattsEvtExchangeMtuRequest(conn_handle, client_mtu)
```

Bases: *GattsEvt*

evt\_id = 85

classmethod from\_c(event)

```
class blatann.nrf.nrf_events.gatt_events.GattsEvtTimeout(conn_handle, source)
```

Bases: *GattsEvt*

evt\_id = 86

classmethod from\_c(event)

### **blatann.nrf.nrf\_events.generic\_events module**

```
class blatann.nrf.nrf_events.generic_events.BLEEvent(conn_handle)
```

Bases: *object*

evt\_id = None

```
class blatann.nrf.nrf_events.generic_events.EvtUserMemoryRequest(conn_handle, request_type)
```

Bases: *BLEEvent*

evt\_id = 1

classmethod from\_c(event)

**blatann.nrf.nrf\_events.smp\_events module**

```
class blatann.nrf.nrf_events.smp_events.GapEvtSec(conn_handle)
```

Bases: [GapEvt](#)

```
class blatann.nrf.nrf_events.smp_events.GapEvtConnSecUpdate(conn_handle, sec_mode, sec_level,  
                                                             encr_key_size)
```

Bases: [GapEvtSec](#)

**evt\_id** = 26

**classmethod** **from\_c**(*event*)

```
class blatann.nrf.nrf_events.smp_events.GapEvtSecInfoRequest(conn_handle, peer_addr, master_id,  
                                                             enc_info, id_info, sign_info)
```

Bases: [GapEvtSec](#)

**evt\_id** = 20

**classmethod** **from\_c**(*event*)

```
class blatann.nrf.nrf_events.smp_events.GapEvtSecRequest(conn_handle, bond, mitm, lesc, keypress)
```

Bases: [GapEvtSec](#)

**evt\_id** = 30

**classmethod** **from\_c**(*event*)

```
class blatann.nrf.nrf_events.smp_events.GapEvtSecParamsRequest(conn_handle, sec_params)
```

Bases: [GapEvtSec](#)

**evt\_id** = 19

**classmethod** **from\_c**(*event*)

```
class blatann.nrf.nrf_events.smp_events.GapEvtAuthKeyRequest(conn_handle, key_type)
```

Bases: [GapEvtSec](#)

**evt\_id** = 23

**classmethod** **from\_c**(*event*)

```
class blatann.nrf.nrf_events.smp_events.GapEvtAuthStatus(conn_handle, auth_status, error_src,  
                                                         bonded, sm1_levels, sm2_levels,  
                                                         kdist_own, kdist_peer)
```

Bases: [GapEvtSec](#)

**evt\_id** = 25

**classmethod** **from\_c**(*event*)

```
class blatann.nrf.nrf_events.smp_events.GapEvtPasskeyDisplay(conn_handle, passkey,  
                                                             match_request)
```

Bases: [GapEvtSec](#)

**evt\_id** = 21

**classmethod** **from\_c**(*event*)

```
class blatann.nrf.nrf_events.smp_events.GapEvtLescDhKeyRequest(conn_handle, remote_public_key,
                                                             oob_required)
```

Bases: [GapEvtSec](#)

evt\_id = 24

classmethod from\_c(event)

## blatann.nrf.nrf\_types package

### Submodules

#### blatann.nrf.nrf\_types.config module

```
class blatann.nrf.nrf_types.config.BleOptionFlag(value)
```

Bases: [IntEnum](#)

An enumeration.

pa\_lna = 1

conn\_event\_extension = 2

gap\_channel\_map = 32

gap\_local\_conn\_latency = 33

gap\_passkey = 34

gap\_scan\_req\_report = 35

gap\_compat\_mode\_1 = 36

gap\_auth\_payload\_timeout = 37

gap\_slave\_latency\_disable = 38

```
class blatann.nrf.nrf_types.config.BleOption
```

Bases: [object](#)

option\_flag = None

path = ''

to\_c()

```
class blatann.nrf.nrf_types.config.BleEnableOpt(enabled=False)
```

Bases: [BleOption](#)

to\_c()

```
class blatann.nrf.nrf_types.config.BleOptConnEventExtension(enabled=False)
```

Bases: [BleEnableOpt](#)

option\_flag = 2

path = 'common\_opt.conn\_evt\_ext'

```

class blatann.nrf.nrf_types.config.BlePaLnaConfig(enabled=False, active_high=True, pin=0)
    Bases: object
    to_c()

class blatann.nrf.nrf_types.config.BleOptPaLna(pa_config=None, lna_cfg=None, ppi_channel_set=0,
                                              ppi_channel_clear=0, gpiote_channel=0)
    Bases: BleOption
    option_flag = 1
    path = 'common_opt.pa_lna'
    to_c()

class blatann.nrf.nrf_types.config.BleOptGapChannelMap(enabled_channels=None, conn_handle=0)
    Bases: BleOption
    option_flag = 32
    path = 'gap_opt.ch_map'
    to_c()

class blatann.nrf.nrf_types.config.BleOptGapLocalConnLatency(conn_handle=0,
                                                             requested_latency=0)
    Bases: BleOption
    option_flag = 33
    path = 'gap_opt.local_conn_latency'
    to_c()

class blatann.nrf.nrf_types.config.BleOptGapPasskey(passkey='000000')
    Bases: BleOption
    option_flag = 34
    path = 'gap_opt.passkey'
    to_c()

class blatann.nrf.nrf_types.config.BleOptGapScanRequestReport(enabled=False)
    Bases: BleEnableOpt
    option_flag = 35
    path = 'gap_opt.scan_req_report'

class blatann.nrf.nrf_types.config.BleOptGapCompatMode1(enabled=False)
    Bases: BleEnableOpt
    option_flag = 36
    path = 'gap_opt.compat_mode_q'

class blatann.nrf.nrf_types.config.BleOptGapAuthPayloadTimeout(conn_handle,
                                                                timeout_ms=48000)
    Bases: BleOption

```

```
    option_flag = 37

    path = 'gap_opt.auth_payload_timeout'

    to_c()

class blatann.nrf.nrf_types.config.BleOptGapSlaveLatencyDisable(conn_handle, disabled=False)
    Bases: BleOption

    option_flag = 38

    path = 'gap_opt.slave_latency_disable'

    to_c()

class blatann.nrf.nrf_types.config.BleEnableConfig(vs_uuid_count=10, periph_role_count=1,
                                                    central_role_count=3, central_sec_count=3,
                                                    service_changed_char=1, attr_table_size=1408)

    Bases: object

    get_vs_uuid_cfg()

    get_role_count_cfg()

    get_device_name_cfg()

    get_service_changed_cfg()

    get_attr_tab_size_cfg()

    get_configs()

class blatann.nrf.nrf_types.config.BleConnConfig(conn_tag=1, conn_count=1, event_length=3,
                                                  write_cmd_tx_queue_size=1, hvn_tx_queue_size=1,
                                                  max_att_mtu=23)

    Bases: object

    DEFAULT_CONN_TAG = 1

    get_gap_config()

    get_gatt_config()

    get_gattc_config()

    get_gatts_config()

    get_configs()
```

### **blatann.nrf.nrf\_types.enums module**

```
class blatann.nrf.nrf_types.enums.BLEHci(value)
    Bases: Enum

    An enumeration.

    success = 0
```



```

unknown_btles_command = 1
unknown_connection_identifier = 2
authentication_failure = 5
pin_or_key_missing = 6
memory_capacity_exceeded = 7
connection_timeout = 8
command_disallowed = 12
invalid_btles_command_parameters = 18
remote_user_terminated_connection = 19
remote_dev_termination_due_to_low_resources = 20
remote_dev_termination_due_to_power_off = 21
local_host_terminated_connection = 22
unsupported_remote_feature = 26
invalid_lmp_parameters = 30
unspecified_error = 31
lmp_response_timeout = 34
lmp_transaction_collision = 35
lmp_pdu_not_allowed = 36
instant_passed = 40
pairing_with_unit_key_unsupported = 41
different_transaction_collision = 42
controller_busy = 58
conn_interval_unacceptable = 59
parameter_out_of_mandatory_range = 48
directed_advertiser_timeout = 60
conn_terminated_due_to_mic_failure = 61
conn_failed_to_be_established = 62
class blatann.nrf.nrf_types.enums.NrfError(value)
    Bases: Enum
    An enumeration.
    success = 0

```

```
svc_handler_missing = 1
softdevice_not_enabled = 2
internal = 3
no_mem = 4
not_found = 5
not_supported = 6
invalid_param = 7
invalid_state = 8
invalid_length = 9
invalid_flags = 10
invalid_data = 11
data_size = 12
timeout = 13
null = 14
forbidden = 15
invalid_addr = 16
busy = 17
conn_count = 18
resources = 19
ble_not_enabled = 12289
ble_invalid_conn_handle = 12290
ble_invalid_attr_handle = 12291
ble_invalid_adv_handle = 12292
ble_invalid_role = 12293
ble_blocked_by_other_links = 12294
ble_gap_uuid_list_mismatch = 12800
ble_gap_discoverable_with_whitelist = 12801
ble_gap_invalid_ble_addr = 12802
ble_gap_whitelist_in_use = 12803
ble_gap_device_identities_in_use = 12804
ble_gap_device_identities_duplicate = 12805
```

```
ble_gattc_proc_not_permitted = 13056
ble_gatts_invalid_attr_type = 13312
ble_gatts_sys_attr_missing = 13313
rpc_encode = 32769
rpc_decode = 32770
rpc_send = 32771
rpc_invalid_argument = 32772
rpc_no_response = 32773
rpc_invalid_state = 32774
rpc_serialization_transport = 32788
rpc_serialization_transport_invalid_state = 32789
rpc_serialization_transport_no_response = 32790
rpc_serialization_transport_already_open = 32791
rpc_serialization_transport_already_closed = 32792
rpc_h5_transport = 32808
rpc_h5_transport_state = 32809
rpc_h5_transport_no_response = 32810
rpc_h5_transport_slip_payload_size = 32811
rpc_h5_transport_slip_calculated_payload_size = 32812
rpc_h5_transport_slip_decoding = 32813
rpc_h5_transport_header_checksum = 32814
rpc_h5_transport_packet_checksum = 32815
rpc_h5_transport_already_open = 32816
rpc_h5_transport_already_closed = 32817
rpc_h5_transport_internal_error = 32818
rpc_serial_port = 32828
rpc_serial_port_state = 32829
rpc_serial_port_already_open = 32830
rpc_serial_port_already_closed = 32831
rpc_serial_port_internal_error = 32832
```

```
class blatann.nrf.nrf_types.enums.BLEGapAdvType(value)
```

Bases: `IntEnum`

An enumeration.

```
connectable_undirected = 0
```

```
connectable_directed = 1
```

```
scanable_undirected = 2
```

```
non_connectable_undirected = 3
```

```
scan_response = 4
```

```
class blatann.nrf.nrf_types.enums.BLEGapRoles(value)
```

Bases: `IntEnum`

An enumeration.

```
invalid = 0
```

```
periph = 1
```

```
central = 2
```

```
class blatann.nrf.nrf_types.enums.BLEGapTimeoutSrc(value)
```

Bases: `IntEnum`

An enumeration.

```
advertising = 0
```

```
scan = 1
```

```
conn = 2
```

```
class blatann.nrf.nrf_types.enums.BLEGapPhy(value)
```

Bases: `IntFlag`

An enumeration.

```
auto = 0
```

```
one_mbps = 1
```

```
two_mbps = 2
```

```
coded = 4
```

```
class blatann.nrf.nrf_types.enums.BLEGapIoCaps(value)
```

Bases: `IntEnum`

An enumeration.

```
DISPLAY_ONLY = 0
```

```
DISPLAY_YESNO = 1
```

```
KEYBOARD_ONLY = 2
```

```
NONE = 3
```

```
KEYBOARD_DISPLAY = 4
```

```
class blatann.nrf.nrf_types.enums.BLEGapAuthKeyType(value)
```

```
    Bases: IntEnum
```

```
    An enumeration.
```

```
    NONE = 0
```

```
    OOB = 2
```

```
    PASSKEY = 1
```

```
class blatann.nrf.nrf_types.enums.BLEGapSecStatus(value)
```

```
    Bases: IntEnum
```

```
    An enumeration.
```

```
    success = 0
```

```
    timeout = 1
```

```
    pdu_invalid = 2
```

```
    passkey_entry_failed = 129
```

```
    oob_not_available = 130
```

```
    auth_req = 131
```

```
    confirm_value = 132
```

```
    pairing_not_supp = 133
```

```
    enc_key_size = 134
```

```
    smp_cmd_unsupported = 135
```

```
    unspecified = 136
```

```
    repeated_attempts = 137
```

```
    invalid_params = 138
```

```
    dhkey_failure = 139
```

```
    num_comp_failure = 140
```

```
    br_edr_in_prog = 141
```

```
    x_trans_key_disallowed = 142
```

```
class blatann.nrf.nrf_types.enums.BLEGattWriteOperation(value)
```

```
    Bases: Enum
```

```
    An enumeration.
```

```
    invalid = 0
```

```
    write_req = 1
```

```
write_cmd = 2
signed_write_cmd = 3
prepare_write_req = 4
execute_write_req = 5
```

```
class blatann.nrf.nrf_types.enums.BLEGattHVXType(value)
```

Bases: [Enum](#)

An enumeration.

```
invalid = 0
notification = 1
indication = 2
```

```
class blatann.nrf.nrf_types.enums.BLEGattStatusCode(value)
```

Bases: [Enum](#)

An enumeration.

```
success = 0
unknown = 1
invalid = 256
invalid_handle = 257
read_not_permitted = 258
write_not_permitted = 259
invalid_pdu = 260
insuf_authentication = 261
request_not_supported = 262
invalid_offset = 263
insuf_authorization = 264
prepare_queue_full = 265
attribute_not_found = 266
attribute_not_long = 267
insuf_enc_key_size = 268
invalid_att_val_length = 269
unlikely_error = 270
insuf_encryption = 271
unsupported_group_type = 272
```

```
insuf_resources = 273
rfu_range1_begin = 274
rfu_range1_end = 383
app_begin = 384
app_end = 415
rfu_range2_begin = 416
rfu_range2_end = 479
rfu_range3_begin = 480
rfu_range3_end = 508
cps_cccd_config_error = 509
cps_proc_alr_in_prog = 510
cps_out_of_range = 511
```

```
class blatann.nrf.nrf_types.enums.BLEGattExecWriteFlag(value)
```

Bases: [Enum](#)

An enumeration.

```
prepared_cancel = 0
prepared_write = 1
unused = 0
```

```
class blatann.nrf.nrf_types.enums.BLEGattsWriteOperation(value)
```

Bases: [Enum](#)

An enumeration.

```
invalid = 0
write_req = 1
write_cmd = 2
sign_write_cmd = 3
prep_write_req = 4
exec_write_req_cancel = 5
exec_write_req_now = 6
```

**blatann.nrf.nrf\_types.gap module**

```
class blatann.nrf.nrf_types.gap.TimeRange(name, val_min, val_max, unit_ms_conversion, divisor=1.0,
                                           units='ms')
```

Bases: `object`

property name: `str`

property min: `float`

property max: `float`

property units: `str`

is\_in\_range(value)

validate(value)

```
class blatann.nrf.nrf_types.gap.BLEGapAdvParams(interval_ms, timeout_s, advertising_type=BLEGapAdvType.connectable_undirected,
                                                  channel_mask=None)
```

Bases: `object`

to\_c()

```
class blatann.nrf.nrf_types.gap.BLEGapScanParams(interval_ms, window_ms, timeout_s, active=True)
```

Bases: `object`

to\_c()

```
class blatann.nrf.nrf_types.gap.BLEGapConnParams(min_conn_interval_ms, max_conn_interval_ms,
                                                  conn_sup_timeout_ms, slave_latency)
```

Bases: `object`

validate()

classmethod from\_c(conn\_params)

to\_c()

```
class blatann.nrf.nrf_types.gap.BLEGapAddrTypes(value)
```

Bases: `IntEnum`

An enumeration.

public = 0

random\_static = 1

random\_private\_resolvable = 2

random\_private\_non\_resolvable = 3

anonymous = 127

```
class blatann.nrf.nrf_types.gap.BLEGapAddr(addr_type, addr)
```

Bases: `object`

classmethod from\_c(addr)



```

    classmethod from_string(addr_string)

    to_c()

    get_addr_type_str()

    get_addr_flag()

class blatann.nrf.nrf_types.gap.BLEAdvData(**kwargs)
    Bases: object
    class Types(value)
        Bases: Enum
        An enumeration.
        flags = 1

        service_16bit_uuid_more_available = 2
        service_16bit_uuid_complete = 3
        service_32bit_uuid_more_available = 4
        service_32bit_uuid_complete = 5
        service_128bit_uuid_more_available = 6
        service_128bit_uuid_complete = 7
        short_local_name = 8
        complete_local_name = 9
        tx_power_level = 10
        class_of_device = 13
        simple_pairing_hash_c = 14
        simple_pairing_randimizer_r = 15
        security_manager_tk_value = 16
        security_manager_oob_flags = 17
        slave_connection_interval_range = 18
        solicited_sevice_uuids_16bit = 20
        solicited_sevice_uuids_128bit = 21
        service_data = 22
        public_target_address = 23
        random_target_address = 24
        appearance = 25
        advertising_interval = 26

```

```
le_bluetooth_device_address = 27
le_role = 28
simple_pairng_hash_c256 = 29
simple_pairng_randomizer_r256 = 30
service_data_32bit_uuid = 32
service_data_128bit_uuid = 33
uri = 36
information_3d_data = 61
manufacturer_specific_data = 255

to_list()
to_c()

classmethod from_c(adv_report_evt)

class blatann.nrf.nrf_types.gap.BLEGapDataLengthParams(max_tx_octets=0, max_rx_octets=0,
                                                         max_tx_time_us=0, max_rx_time_us=0)
    Bases: object
    to_c()

class blatann.nrf.nrf_types.gap.BLEGapPhys(tx_phys=BLEGapPhy.auto, rx_phys=BLEGapPhy.auto)
    Bases: object
    to_c()

class blatann.nrf.nrf_types.gap.BLEGapPrivacyParams(enabled=False, resolvable_addr=False,
                                                      addr_update_rate_s=900)
    Bases: object
    DEFAULT_PRIVATE_ADDR_CYCLE_INTERVAL_S = 900
    to_c()
    classmethod from_c(privacy)
```

### blatann.nrf.nrf\_types.gatt module

```
blatann.nrf.nrf_types.gatt.BLE_GATT_HANDLE_INVALID = 0
    GATT Classes

class blatann.nrf.nrf_types.gatt.BleGattEnableParams(max_att_mtu=0)
    Bases: object
    to_c()
```

```
class blatann.nrf.nrf_types.gatt.BLEGattCharacteristicProperties(broadcast=False, read=False,
                                                                write_wo_resp=False,
                                                                write=False, notify=False,
                                                                indicate=False,
                                                                auth_signed_wr=False)
```

Bases: `object`

`classmethod from_c(gattc_char_props)`

`to_c()`

```
class blatann.nrf.nrf_types.gatt.BLEGattExtendedCharacteristicProperties(reliable_write=False,
                                                                           writable_aux=False)
```

Bases: `object`

`to_c()`

`classmethod from_c(params)`

```
class blatann.nrf.nrf_types.gatt.BLEGattService(uuid, start_handle, end_handle)
```

Bases: `object`

`srvc_uuid = 0x2800` (Standard.service\_primary)

`classmethod from_c(gattc_service)`

`char_add(char)`

```
class blatann.nrf.nrf_types.gatt.BLEGattCharacteristic(uuid, handle_decl, handle_value,
                                                         data_decl=None, data_value=None,
                                                         char_props=None)
```

Bases: `object`

`char_uuid = 0x2803` (Standard.characteristic)

`discovered_handles()`

`missing_handles()`

`classmethod from_c(gattc_char)`

```
class blatann.nrf.nrf_types.gatt.BleGattHandle(handle=0)
```

Bases: `object`

```
class blatann.nrf.nrf_types.gatt.BLEGattcWriteParams(write_op, flags, handle, data, offset)
```

Bases: `object`

`classmethod from_c(gattc_write_params)`

`to_c()`

```
class blatann.nrf.nrf_types.gatt.BLEGattcDescriptor(uuid, handle, data=None)
```

Bases: `object`

`classmethod from_c(gattc_desc)`

```
class blatann.nrf.nrf_types.gatt.BLEGattcAttrInfo16(handle, uuid)
```

Bases: `object`

```
    classmethod from_c(attr_info16)

class blatann.nrf.nrf_types.gatt.BLEGattcAttrInfo128(attr_handle, uuid)
    Bases: object
    classmethod from_c(attr_info128)

class blatann.nrf.nrf_types.gatt.BLEGattsEnableParams(service_changed, attribute_table_size)
    Bases: object
    to_c()

class blatann.nrf.nrf_types.gatt.BLEGattsCharHandles(value_handle=0, user_desc_handle=0,
                                                    cccd_handle=0, sccd_handle=0)
    Bases: object
    to_c()
    classmethod from_c(handle_params)

class blatann.nrf.nrf_types.gatt.BLEGattsAttribute(uuid, attr_metadata, max_len, value=b'')
    Bases: object
    to_c()

class blatann.nrf.nrf_types.gatt.BLEGattsPresentationFormat(fmt, exponent, unit, namespace,
                                                            description)
    Bases: object
    to_c()
    classmethod from_c(params)

class blatann.nrf.nrf_types.gatt.BLEGattsAttrMetadata(read_permissions=<blatann.nrf.nrf_types.smp.BLEGapSecMod
                                                    object>,
                                                    write_permissions=<blatann.nrf.nrf_types.smp.BLEGapSecMod
                                                    object>, variable_length=False,
                                                    read_auth=False, write_auth=False)
    Bases: object
    to_c()
    classmethod from_c(params)

class blatann.nrf.nrf_types.gatt.BLEGattsCharMetadata(char_props, user_description="",
                                                    user_description_max_size=0,
                                                    user_desc_metadata=None,
                                                    cccd_metadata=None, sccd_metadata=None,
                                                    presentation_format=None)
    Bases: object
    to_c()
    classmethod from_c(params)

class blatann.nrf.nrf_types.gatt.BLEGattsAuthorizeParams(gatt_status, update, offset=0, data="")
    Bases: object
```

```
to_c()
```

```
class blatann.nrf.nrf_types.gatt.BLEGattsRwAuthorizeReplyParams(read=None, write=None)
```

```
Bases: object
```

```
to_c()
```

```
class blatann.nrf.nrf_types.gatt.BLEGattsValue(value, offset=0)
```

```
Bases: object
```

```
to_c()
```

```
classmethod from_c(params)
```

```
class blatann.nrf.nrf_types.gatt.BLEGattsHvx(char_handle, hvx_type, data, offset=0)
```

```
Bases: object
```

```
to_c()
```

### blatann.nrf.nrf\_types.generic module

```
class blatann.nrf.nrf_types.generic.BLEUUIDBase(vs_uuid_base=None, uuid_type=None)
```

```
Bases: object
```

```
BLE_UUID_TYPE_BLE = 1
```

```
classmethod from_c(uuid)
```

```
classmethod from_uuid128_array(uuid128_array)
```

```
to_c()
```

```
class blatann.nrf.nrf_types.generic.BLEUUID(value,
                                             base=<blatann.nrf.nrf_types.generic.BLEUUIDBase
                                             object>)
```

```
Bases: object
```

```
class Standard(value)
```

```
Bases: Enum
```

```
An enumeration.
```

```
unknown = 0
```

```
service_primary = 10240
```

```
service_secondary = 10241
```

```
characteristic = 10243
```

```
cccd = 10498
```

```
battery_level = 10777
```

```
heart_rate = 10807
```

```
get_value()
```

```
as_array()
classmethod from_c(uuid)
classmethod from_uuid128(uuid128)
to_c()
classmethod from_array(uuid_array_lt)
```

### blatann.nrf.nrf\_types.smp module

```
class blatann.nrf.nrf_types.smp.BLEGapSecMode(sec_mode, level)
    Bases: object
    to_c()
    classmethod from_c(params)

class blatann.nrf.nrf_types.smp.BLEGapSecModeType
    Bases: object
    NO_ACCESS = <blatann.nrf.nrf_types.smp.BLEGapSecMode object>
    OPEN = <blatann.nrf.nrf_types.smp.BLEGapSecMode object>
    ENCRYPTION = <blatann.nrf.nrf_types.smp.BLEGapSecMode object>
    MITM = <blatann.nrf.nrf_types.smp.BLEGapSecMode object>
    LESC_MITM = <blatann.nrf.nrf_types.smp.BLEGapSecMode object>
    SIGN_OR_ENCRYPT = <blatann.nrf.nrf_types.smp.BLEGapSecMode object>
    SIGN_OR_ENCRYPT_MITM = <blatann.nrf.nrf_types.smp.BLEGapSecMode object>

class blatann.nrf.nrf_types.smp.BLEGapSecLevels(lv1, lv2, lv3, lv4)
    Bases: object
    classmethod from_c(sec_level)
    to_c()

class blatann.nrf.nrf_types.smp.BLEGapSecKeyDist(enc_key=False, id_key=False, sign_key=False,
                                                  link_key=False)
    Bases: object
    classmethod from_c(kdist)
    to_c()

class blatann.nrf.nrf_types.smp.BLEGapSecParams(bond, mitm, le_sec_pairing, keypress_noti, io_caps,
                                                  oob, min_key_size, max_key_size, kdist_own,
                                                  kdist_peer)
    Bases: object
    classmethod from_c(sec_params)
```

```

    to_c()
class blatann.nrf.nrf_types.smp.BLEGapMasterId(ediv=0, rand=b'')
    Bases: object
    RAND_LEN = 8
    RAND_INVALID = b'\x00\x00\x00\x00\x00\x00\x00\x00'
    to_c()
    property is_valid: bool
    classmethod from_c(master_id)
    to_dict()
    classmethod from_dict(data)
class blatann.nrf.nrf_types.smp.BLEGapEncryptInfo(ltk=b'', lesc=False, auth=False)
    Bases: object
    KEY_LENGTH = 16
    to_c()
    classmethod from_c(info)
    to_dict()
    classmethod from_dict(data)
class blatann.nrf.nrf_types.smp.BLEGapEncryptKey(enc_info=None, master_id=None)
    Bases: object
    to_c()
    classmethod from_c(key)
    to_dict()
    classmethod from_dict(data)
class blatann.nrf.nrf_types.smp.BLEGapIdKey(irk=b'', peer_addr=None)
    Bases: object
    KEY_LENGTH = 16
    to_c()
    classmethod from_c(id_key)
    to_dict()
    classmethod from_dict(data)
class blatann.nrf.nrf_types.smp.BLEGapPublicKey(key=b'')
    Bases: object
    KEY_LENGTH = 64

```

```
    to_c()

    classmethod from_c(key)

class blatann.nrf.nrf_types.smp.BLEGapDhKey(key=b'')
    Bases: object
    KEY_LENGTH = 32
    to_c()
    classmethod from_c(key)

class blatann.nrf.nrf_types.smp.BLEGapSignKey(key=b'')
    Bases: object
    KEY_LENGTH = 16
    to_c()
    classmethod from_c(key)
    to_dict()
    classmethod from_dict(data)

class blatann.nrf.nrf_types.smp.BLEGapSecKeys(enc_key=None, id_key=None, sign_key=None,
                                              public_key=None)
    Bases: object
    to_c()
    classmethod from_c(keys)

class blatann.nrf.nrf_types.smp.BLEGapSecKeyset(own_keys=None, peer_keys=None)
    Bases: object
    to_c()
    reload()
    classmethod from_c(keyset)
```

## Submodules

### blatann.nrf.nrf\_dll\_load module

### blatann.nrf.nrf\_driver module

```
blatann.nrf.nrf_driver.NordicSemiErrorCheck(wrapped=None, expected=0)
```

```
class blatann.nrf.nrf_driver.NrfDriverObserver
```

```
    Bases: object
```

```
    on_driver_event(nrf_driver, event)
```



---

```

class blatann.nrf.nrf_driver.NrfDriver(serial_port, baud_rate=None, log_driver_comms=False)
    Bases: object
    default_baud_rate = 1000000
    ATT_MTU_DEFAULT = 23
    property serial_port
    open()
    property is_open
    close()
    event_subscribe(handler, *event_types)
    event_unsubscribe(handler, *event_types)
    event_unsubscribe_all(handler)
    observer_register(observer)
    observer_unregister(observer)
    ble_enable_params_setup()
    adv_params_setup()
    scan_params_setup()
    conn_params_setup()
    security_params_setup()
    ble_conn_configure(conn_params)
    ble_enable(ble_enable_params=None)
    ble_opt_set(ble_opt)
    ble_user_mem_reply(conn_handle)
    ble_vs_uuid_add(uuid_base)
    ble_gap_addr_get()
    ble_gap_addr_set(address)
    ble_gap_device_name_set(name)
    ble_gap_appearance_set(value)
    ble_gap_ppcp_set(conn_params)
    ble_gap_tx_power_set(tx_power)
    ble_gap_privacy_set(privacy)
    ble_gap_adv_start(adv_params=None, conn_cfg_tag=0)

```

```
ble_gap_conn_param_update(conn_handle, conn_params)

ble_gap_adv_stop()

ble_gap_scan_start(scan_params=None)

ble_gap_scan_stop()

ble_gap_rssi_start(conn_handle, threshold_dbm, skip_count)

ble_gap_rssi_stop(conn_handle)

ble_gap_rssi_get(conn_handle)

ble_gap_connect(address, scan_params=None, conn_params=None, conn_cfg_tag=0)

ble_gap_disconnect(conn_handle, hci_status_code=BLEHci.remote_user_terminated_connection)

ble_gap_adv_data_set(adv_data={}, scan_data={})

ble_gap_data_length_update(conn_handle, params=None)

ble_gap_phy_update(conn_handle, tx_phy=BLEGapPhy.auto, rx_phy=BLEGapPhy.auto)

ble_gap_authenticate(conn_handle, sec_params)

ble_gap_sec_params_reply(conn_handle, sec_status, sec_params, sec_keyset)

ble_gap_auth_key_reply(conn_handle, key_type, key)

ble_gap_sec_info_reply(conn_handle, enc_info=None, irk=None, sign_info=None)

ble_gap_encrypt(conn_handle, master_id, enc_info)

ble_gap_lesc_dhkey_reply(conn_handle, dh_key)

ble_gatts_service_add(service_type, uuid, service_handle)

ble_gatts_characteristic_add(service_handle, char_md, attr_char_value, char_handle)

ble_gatts_descriptor_add(char_handle, attr)

ble_gatts_rw_authorize_reply(conn_handle, authorize_reply_params)

ble_gatts_value_get(conn_handle, attribute_handle, gatts_value, max_bytes_read=512)

ble_gatts_value_set(conn_handle, attribute_handle, gatts_value)

ble_gatts_hvx(conn_handle, hvx_params)

ble_gatts_service_changed(conn_handle, start_handle, end_handle)

ble_gatts_exchange_mtu_reply(conn_handle, server_mtu)

ble_gatts_sys_attr_set(conn_handle, sys_attr_data, flags=0)

ble_gattc_write(conn_handle, write_params)

ble_gattc_prim_srvc_disc(conn_handle, srvc_uuid, start_handle)

ble_gattc_char_disc(conn_handle, start_handle, end_handle)
```

```

ble_gattc_desc_disc(conn_handle, start_handle, end_handle)

ble_gattc_attr_info_disc(conn_handle, start_handle, end_handle)

ble_gattc_read(conn_handle, read_handle, offset=0)

ble_gattc_exchange_mtu_req(conn_handle, att_mtu_size)

ble_gattc_hv_confirm(conn_handle, attr_handle)

ble_evt_handler(adapter, ble_event)

```

### blatann.nrf.nrf\_driver\_types module

```

blatann.nrf.nrf_driver_types.msec_to_units(time_ms, resolution)
    Convert milliseconds to BLE specific time units.

blatann.nrf.nrf_driver_types.units_to_msec(units, resolution)
    Convert BLE specific units to milliseconds.

blatann.nrf.nrf_driver_types.char_array_to_list(array_pointer, length)
    Convert char_array to python list.

blatann.nrf.nrf_driver_types.uint8_array_to_list(array_pointer, length)
    Convert uint8_array to python list.

blatann.nrf.nrf_driver_types.uint16_array_to_list(array_pointer, length)
    Convert uint16_array to python list.

blatann.nrf.nrf_driver_types.service_array_to_list(array_pointer, length)
    Convert ble_gattc_service_array to python list.

blatann.nrf.nrf_driver_types.include_array_to_list(array_pointer, length)
    Convert ble_gattc_include_array to python list.

blatann.nrf.nrf_driver_types.ble_gattc_char_array_to_list(array_pointer, length)
    Convert ble_gattc_char_array to python list.

blatann.nrf.nrf_driver_types.desc_array_to_list(array_pointer, length)
    Convert ble_gattc_desc_array to python list.

blatann.nrf.nrf_driver_types.ble_gattc_attr_info16_array_to_list(array_pointer, length)
    Convert ble_gattc_attr_info16_array to python list

blatann.nrf.nrf_driver_types.ble_gattc_attr_info128_array_to_list(array_pointer, length)
    Convert ble_gattc_attr_info128_array to python list

blatann.nrf.nrf_driver_types.handle_value_array_to_list(array_pointer, length)
    Convert ble_gattc_handle_value_array to python list.

blatann.nrf.nrf_driver_types.attr_info_array_to_list(array_pointer, length)
    Convert ble_gattc_attr_info_array to python list.

blatann.nrf.nrf_driver_types.attr_info16_array_to_list(array_pointer, length)
    Convert ble_gattc_attr_info16_array to python list.

```

`blatann.nrf.nrf_driver_types.attr_info128_array_to_list(array_pointer, length)`

Convert `ble_gattc_attr_info128_array` to python list.

`blatann.nrf.nrf_driver_types.serial_port_desc_array_to_list(array_pointer, length)`

Convert `sd_rpc_serial_port_desc_array` to python list.

`blatann.nrf.nrf_driver_types.list_to_char_array(data_list)`

Convert python list to `char_array`.

`blatann.nrf.nrf_driver_types.list_to_uint8_array(data_list)`

Convert python list to `uint8_array`.

`blatann.nrf.nrf_driver_types.list_to_uint16_array(data_list)`

Convert python list to `uint16_array`.

`blatann.nrf.nrf_driver_types.list_to_service_array(data_list)`

Convert python list to `ble_gattc_service_array`.

`blatann.nrf.nrf_driver_types.list_to_include_array(data_list)`

Convert python list to `ble_gattc_include_array`.

`blatann.nrf.nrf_driver_types.list_to_ble_gattc_char_array(data_list)`

Convert python list to `ble_gattc_char_array`.

`blatann.nrf.nrf_driver_types.list_to_desc_array(data_list)`

Convert python list to `ble_gattc_desc_array`.

`blatann.nrf.nrf_driver_types.list_to_handle_value_array(data_list)`

Convert python list to `ble_gattc_handle_value_array`.

`blatann.nrf.nrf_driver_types.list_to_serial_port_desc_array(data_list)`

Convert python list to `sd_rpc_serial_port_desc_array`.

## **blatann.services package**

### **Subpackages**

#### **blatann.services.battery package**

`blatann.services.battery.add_battery_service(gatts_database, enable_notifications=False, security_level=SecurityLevel.OPEN)`

Adds a battery service to the given GATT Server database

##### **Parameters**

- **gatts\_database** (`blatann.gatt.gatts.GattsDatabase`) – The database to add the service to
- **enable\_notifications** – Whether or not the Battery Level characteristic allows notifications
- **security\_level** – The security level to use for the service

##### **Returns**

The Battery service

##### **Return type**

`_BatteryServer`

`blatann.services.battery.find_battery_service(gattc_database)`

Finds a battery service in the given GATT client database

**Parameters**

**gattc\_database** (`blatann.gatt.gattc.GattcDatabase`) – the GATT client database to search

**Returns**

The Battery service if found, None if not found

**Return type**

`_BatteryClient`

## Submodules

`blatann.services.battery.constants` module

`blatann.services.battery.data_types` module

`class blatann.services.battery.data_types.BatteryLevel`

Bases: `UInt8`

`blatann.services.battery.service` module

`class blatann.services.battery.service.BatteryServer(service, enable_notifications=False, security_level=SecurityLevel.OPEN)`

Bases: `object`

**set\_battery\_level**(*battery\_percent*, *notify\_client=True*)

Sets the new battery level in the service

**Parameters**

- **battery\_percent** (`int`) – The new battery percent
- **notify\_client** – Whether or not to notify the connected client with the updated value

**classmethod add\_to\_database**(*gatts\_database*, *enable\_notifications=False*, *security\_level=SecurityLevel.OPEN*)

`class blatann.services.battery.service.BatteryClient(gattc_service)`

Bases: `object`

**read()**

Reads the Battery level characteristic.

**Return type**

`EventWaitable[BatteryClient, DecodedReadCompleteEventArgs[int]]`

**Returns**

A waitable for when the read completes, which waits for the `on_battery_level_update_event` to be emitted

**property on\_battery\_level\_updated:** *Event[BatteryClient, DecodedReadCompleteEventArgs[int]]*

Event that is generated whenever the battery level on the peripheral is updated, whether it is by notification or from reading the characteristic itself.

The DecodedReadCompleteEventArgs value given is the integer battery percent received. If the read failed or failed to decode, the value will be equal to the raw bytes received.

**property can\_enable\_notifications:** *bool*

Checks if the battery level characteristic allows notifications to be subscribed to

**Returns**

True if notifications can be enabled, False if not

**enable\_notifications()**

Enables notifications for the battery level characteristic. Note: this function will raise an exception if notifications aren't possible

**Returns**

a Waitable which waits for the write to finish

**disable\_notifications()**

Disables notifications for the battery level characteristic. Note: this function will raise an exception if notifications aren't possible

**Returns**

a Waitable which waits for the write to finish

**classmethod find\_in\_database(gattc\_database)**

**Return type**

*BatteryClient*

## blatann.services.current\_time package

**blatann.services.current\_time.add\_current\_time\_service(gatts\_database, enable\_writes=False, enable\_local\_time\_info=False, enable\_reference\_info=False)**

Adds a Current Time service to the given GATT server database

**Parameters**

- **gatts\_database** (*blatann.gatt.gatts.GattsDatabase*) – The database to add the service to
- **enable\_writes** – Makes the Current time and Local Time info characteristics writable so clients/centrals can update the server's time
- **enable\_local\_time\_info** – Enables the Local Time characteristic in the service
- **enable\_reference\_info** – Enables the Reference Info characteristic in the service

**Returns**

The Current Time service

**Return type**

*\_CurrentTimeServer*

## Submodules

`blatann.services.current_time.constants` module

`blatann.services.current_time.data_types` module

`class blatann.services.current_time.data_types.DaylightSavingsTimeOffset(value)`

Bases: `IntEnum`

An enumeration.

`standard_time = 0`

`half_hour_dst = 2`

`full_hour_dst = 4`

`two_hour_dst = 8`

`unknown = 255`

`static from_seconds(seconds)`

Converts the DST offset in seconds to one of the above enums. Values which do not map directly to an above enum will be mapped to unknown. Valid values are essentially 0, 1800 (1/2 hr), 3600 (1 hr), and 7200 (2 hr)

### Parameters

**seconds** – DST offset in seconds

### Returns

The corresponding enum value

`class blatann.services.current_time.data_types.AdjustmentReasonType(value)`

Bases: `IntEnum`

An enumeration.

`manual_time_update = 0`

`external_time_reference_update = 1`

`time_zone_change = 2`

`dst_change = 3`

`class blatann.services.current_time.data_types.TimeSource(value)`

Bases: `IntEnum`

An enumeration.

`unknown = 0`

`network_time_protocol = 1`

`gps = 2`

`radio_time_signal = 3`

`manual = 4`

```
atomic_clock = 5
```

```
cellular_network = 6
```

```
class blatann.services.current_time.data_types.TimeAccuracy(value)
```

Bases: [IntEnum](#)

An enumeration.

```
out_of_range = 254
```

```
unknown = 255
```

```
class blatann.services.current_time.data_types.AdjustmentReason(*adjustment_reasons)
```

Bases: [Bitfield](#)

```
bitfield_width = 8
```

```
bitfield_enum
```

alias of [AdjustmentReasonType](#)

```
class blatann.services.current_time.data_types.ExactTime256(date)
```

Bases: [BleCompoundDataType](#)

```
data_stream_types = [<class 'blatann.services.ble_data_types.DayDateTime'>, <class  
'blatann.services.ble_data_types.Uint8'>]
```

```
encode()
```

**Return type**

[BleDataStream](#)

```
classmethod decode(stream)
```

**Returns**

The values decoded from the stream

**Return type**

[tuple](#)

```
class blatann.services.current_time.data_types.CurrentTime(date, adjustment_reason=None)
```

Bases: [BleCompoundDataType](#)

Class used to report the current time and reason for adjustment

```
data_stream_types = [<class  
'blatann.services.current_time.data_types.ExactTime256'>, <class  
'blatann.services.current_time.data_types.AdjustmentReason'>]
```

```
encode()
```

**Return type**

[BleDataStream](#)

```
classmethod decode(stream)
```

**Returns**

The values decoded from the stream

**Return type**

[tuple](#)



```
class blatann.services.current_time.data_types.LocalTimeInfo(
    timezone_offset_hrs=0.0,
    dst_offset=DaylightSavingsTimeOffset.standard_time)
```

Bases: *BleCompoundDataType*

```
data_stream_types = [<class 'blatann.services.ble_data_types.Int8'>, <class
'blatann.services.ble_data_types.Uint8'>]
```

```
encode()
```

**Return type**

*BleDataStream*

```
classmethod decode(stream)
```

**Returns**

The values decoded from the stream

**Return type**

*tuple*

```
class blatann.services.current_time.data_types.ReferenceTimeInfo(
    source=TimeSource.unknown,
    accu-
    racy_seconds=TimeAccuracy.unknown,
    hours_since_update=None)
```

Bases: *BleCompoundDataType*

```
data_stream_types = [<class 'blatann.services.ble_data_types.Uint8'>, <class
'blatann.services.ble_data_types.Uint8'>, <class
'blatann.services.ble_data_types.Uint8'>, <class
'blatann.services.ble_data_types.Uint8'>]
```

```
encode()
```

**Return type**

*BleDataStream*

```
classmethod decode(stream)
```

**Returns**

The values decoded from the stream

**Return type**

*tuple*

## blatann.services.current\_time.service module

```
class blatann.services.current_time.service.CurrentTimeServer(
    service, is_writable=False, en-
    able_local_time_info_char=False,
    enable_ref_time_info_char=False)
```

Bases: *object*

**property is\_writable: bool**

Gets whether or not the service was configured to allow writes to the Current Time and Local Time Info characteristics

**property has\_local\_time\_info: bool**

Gets whether or not the service was configured to show the Local Time Info characteristic

**property has\_reference\_time\_info:** `bool`

Gets whether or not the service was configured to show the Reference Time Info characteristic

**property on\_current\_time\_write:** `Event[CurrentTimeServer, DecodedWriteEventArgs[CurrentTime]]`

Event that is triggered when a client writes to the Current Time Characteristic. Event emits a DecodedWriteEventArgs argument where the value is of type `current_time.CurrentTime`

**property on\_local\_time\_info\_write:** `Event[CurrentTimeServer, DecodedWriteEventArgs[LocalTimeInfo]]`

Event that is triggered when a client writes to the Local Time Info Characteristic (if present). Event emits a DecodedWriteEventArgs argument where the value is of type `current_time.LocalTimeInfo`

**configure\_automatic()**

Configures the current time and local time info (if present) to use the system time

**set\_time**(*date=None, adjustment\_reason=None, characteristic\_read\_callback=None*)

Manually sets the time to report to the client.

If `characteristic_read_callback` is supplied, the function is called for future reads on that characteristic to fetch the current time. If `characteristic_read_callback` is `None`, future reads will be based off of the base datetime given and the time passed

#### Parameters

- **date** (`datetime.datetime`) – The new base date to set the characteristic to. Future characteristic reads will base its time off of this value if `characteristic_read_callback` is not supplied. If the date is not supplied, will use the current system time (same as `configure_automatic` but doesn't configure local time info)
- **adjustment\_reason** (`AdjustmentReason`) – Optional reason to give for the adjustment
- **characteristic\_read\_callback** – Optional callback to fetch subsequent time values. Function signature should take no parameters and return a datetime object

**set\_local\_time\_info**(*timezone\_hrs=0.0, dst\_offset=DaylightSavingsTimeOffset.standard\_time*)

Sets the local time info characteristic data. Only valid if `has_local_time_info` is `True`

#### Parameters

- **timezone\_hrs** – The timezone to report, in hours
- **dst\_offset** (`DaylightSavingsTimeOffset`) – The daylight savings time offset

#### Raises

`InvalidOperationException` if the service was not configured with the local time info

**set\_reference\_info**(*time\_source=TimeSource.unknown, accuracy=TimeAccuracy.unknown, hours\_since\_update=None*)

Sets the time reference info characteristic data. Only valid if `has_reference_time_info` is `True`

#### Parameters

- **time\_source** (`TimeSource`) – The time source to use
- **accuracy** (`TimeAccuracy`) – The accuracy to report
- **hours\_since\_update** – The number of hours since time reference has been updated

#### Raises

`InvalidOperationException` if the service was not configured with the reference info

```
classmethod add_to_database(gatts_database, is_writable=False, enable_local_time_info_char=False,
                           enable_ref_time_info_char=False)
```

```
class blatann.services.current_time.service.CurrentTimeClient(gattc_service)
```

Bases: `object`

```
property on_current_time_updated: Event[CurrentTimeClient,
DecodedReadCompleteEventArgs[CurrentTime]]
```

Event triggered when the server has updated its current time

```
property on_local_time_info_updated: Event[CurrentTimeClient,
DecodedReadCompleteEventArgs[LocalTimeInfo]]
```

Event triggered when the server has updated its local time info

```
property on_reference_info_updated: Event[CurrentTimeClient,
DecodedReadCompleteEventArgs[ReferenceTimeInfo]]
```

Event triggered when the server has updated its reference time info

```
property has_local_time_info: bool
```

```
property has_reference_info: bool
```

```
property can_enable_notifications: bool
```

```
property can_set_current_time: bool
```

```
property can_set_local_time_info: bool
```

```
read_time()
```

Reads the time from the server

**Return type**

```
EventWaitable[CurrentTimeClient, DecodedReadCompleteEventArgs[CurrentTime]]
```

```
set_time(date, adjustment_reason=None)
```

Sets the time on the server to the datetime provided

## blatann.services.device\_info package

```
blatann.services.device_info.add_device_info_service(gatts_database)
```

**Return type**

```
_DisServer
```

```
blatann.services.device_info.find_device_info_service(gattc_database)
```

**Return type**

```
_DisClient
```

## Submodules

`blatann.services.device_info.constants` module

`blatann.services.device_info.data_types` module

**class** `blatann.services.device_info.data_types.PnpVendorSource(value)`

Bases: `IntEnum`

An enumeration.

`bluetooth_sig` = 1

`usb_vendor` = 2

**class** `blatann.services.device_info.data_types.PnpId(vendor_id_source, vendor_id, product_id, product_revision)`

Bases: `BleCompoundDataType`

`data_stream_types` = [`<class 'blatann.services.ble_data_types.Uint8'>`, `<class 'blatann.services.ble_data_types.Uint16'>`, `<class 'blatann.services.ble_data_types.Uint16'>`, `<class 'blatann.services.ble_data_types.Uint16'>`]

`encode()`

**Return type**

`BleDataStream`

**classmethod** `decode(stream)`

**Returns**

The values decoded from the stream

**Return type**

`tuple`

**class** `blatann.services.device_info.data_types.SystemId(manufacturer_id, organizationally_unique_id)`

Bases: `BleCompoundDataType`

`data_stream_types` = [`<class 'blatann.services.ble_data_types.Uint40'>`, `<class 'blatann.services.ble_data_types.Uint24'>`]

`encode()`

**Return type**

`ble_data_types.BleDataStream`

**classmethod** `decode(stream)`

**Returns**

The values decoded from the stream

**Return type**

`tuple`

**blatann.services.device\_info.service module**

```
class blatann.services.device_info.service.DisClient(gattc_service)
```

```
    Bases: _DeviceInfoService
```

```
    get(characteristic)
```

```
    get_system_id()
```

```
    get_model_number()
```

```
    get_serial_number()
```

```
    get_firmware_revision()
```

```
    get_hardware_revision()
```

```
    get_software_revision()
```

```
    get_manufacturer_name()
```

```
    get_regulatory_certifications()
```

```
    get_pnp_id()
```

```
    classmethod find_in_database(gattc_database)
```

```
        Return type
```

```
        DisClient
```

```
class blatann.services.device_info.service.DisServer(service)
```

```
    Bases: _DeviceInfoService
```

```
    set(characteristic, value, max_len=None)
```

```
    set_system_id(system_id)
```

```
    set_model_number(model_number, max_len=None)
```

```
    set_serial_number(serial_number, max_len=None)
```

```
    set_firmware_revision(firmware_revision, max_len=None)
```

```
    set_hardware_revision(hardware_revision, max_len=None)
```

```
    set_software_revision(software_revision, max_len=None)
```

```
    set_manufacturer_name(manufacturer_name, max_len=None)
```

```
    set_regulatory_certifications(certs)
```

```
    set_pnp_id(pnp_id)
```

```
    classmethod add_to_database(gatts_database)
```

## blatann.services.glucose package

```
blatann.services.glucose.add_glucose_service(gatts_database, glucose_database,  
                                             security_level=SecurityLevel.OPEN,  
                                             include_context_characteristic=True)
```

Adds the Glucose bluetooth service to the Gatt Server database

### Parameters

- **gatts\_database** (`blatann.gatt.gatts.GattsDatabase`) – The GATT database to add the service to
- **glucose\_database** (`IGlucoseDatabase`) – The database which holds the glucose measurements
- **security\_level** (`SecurityLevel`) – The security level for the record-access control point of the service
- **include\_context\_characteristic** – flag whether or not to include the optional context characteristic in the service. If this is False, any context stored with glucose measurements will not be reported.

## Submodules

### blatann.services.glucose.constants module

### blatann.services.glucose.data\_types module

```
class blatann.services.glucose.data_types.GlucoseConcentrationUnits(value)
```

Bases: `IntEnum`

The concentration units available for reporting glucose levels

```
kg_per_liter = 0
```

```
mol_per_liter = 1
```

```
class blatann.services.glucose.data_types.GlucoseType(value)
```

Bases: `IntEnum`

The glucose types available

```
capillary_whole_blood = 1
```

```
capillary_plasma = 2
```

```
venous_whole_blood = 3
```

```
venous_plasma = 4
```

```
arterial_whole_blood = 5
```

```
arterial_plasma = 6
```

```
undetermined_whole_blood = 7
```

```
undetermined_plasma = 8
```

```
interstitial_fluid = 9
```

```
control_solution = 10
```

```
class blatann.services.glucose.data_types.SampleLocation(value)
```

```
    Bases: IntEnum
```

```
    Location which the blood sample was taken
```

```
    finger = 1
```

```
    alternate_test_site = 2
```

```
    earlobe = 3
```

```
    control_solution = 4
```

```
    unknown = 15
```

```
class blatann.services.glucose.data_types.MedicationUnits(value)
```

```
    Bases: IntEnum
```

```
    Available units to report medication values in
```

```
    milligrams = 0
```

```
    milliliters = 1
```

```
class blatann.services.glucose.data_types.CarbohydrateType(value)
```

```
    Bases: IntEnum
```

```
    The type of carbohydrate consumed by the user
```

```
    breakfast = 1
```

```
    lunch = 2
```

```
    dinner = 3
```

```
    snack = 4
```

```
    drink = 5
```

```
    supper = 6
```

```
    brunch = 7
```

```
class blatann.services.glucose.data_types.MealType(value)
```

```
    Bases: IntEnum
```

```
    The type of meal consumed
```

```
    preprandial = 1
```

```
    postprandial = 2
```

```
    fasting = 3
```

```
    casual = 4
```

```
    bedtime = 5
```

```
class blatann.services.glucose.data_types.TesterType(value)
```

Bases: `IntEnum`

Information about who tested the glucose levels

```
self = 1
```

```
health_care_professional = 2
```

```
lab_test = 3
```

```
not_available = 15
```

```
class blatann.services.glucose.data_types.HealthStatus(value)
```

Bases: `IntEnum`

Current health status of the user

```
minor_issues = 1
```

```
major_issues = 2
```

```
during_menses = 3
```

```
under_stress = 4
```

```
normal = 5
```

```
not_available = 15
```

```
class blatann.services.glucose.data_types.MedicationType(value)
```

Bases: `IntEnum`

Medication type consumed

```
rapid_acting_insulin = 1
```

```
short_acting_insulin = 2
```

```
intermediate_acting_insulin = 3
```

```
long_acting_insulin = 4
```

```
premixed_insulin = 5
```

```
class blatann.services.glucose.data_types.SensorStatusType(value)
```

Bases: `IntEnum`

The types of sensor statuses that can be communicated

```
battery_low = 0
```

```
sensor_malfunction = 1
```

```
sample_size_insufficient = 2
```

```
strip_insertion_error = 3
```

```
incorrect_strip_type = 4
```

```
result_above_range = 5
```



```
result_below_range = 6
```

```
sensor_temp_high = 7
```

```
sensor_temp_low = 8
```

```
sensor_read_interrupted = 9
```

```
general_device_fault = 10
```

```
time_fault = 11
```

```
class blatann.services.glucose.data_types.SensorStatus(*sensor_statuses)
```

Bases: *Bitfield*

Class which holds the current sensor status information

```
bitfield_width = 16
```

```
bitfield_enum
```

alias of *SensorStatusType*

```
class blatann.services.glucose.data_types.GlucoseFeatureType(value)
```

Bases: *IntEnum*

Enumeration of the supported feature types to be reported using the Feature characteristic

```
low_battery_detection = 0
```

```
sensor_malfunction_detection = 1
```

```
sensor_sample_size = 2
```

```
strip_insertion_error_detection = 3
```

```
strip_type_error_detection = 4
```

```
sensor_result_high_low_detection = 5
```

```
sensor_temp_high_low_detection = 6
```

```
sensor_read_interrupt_detection = 7
```

```
general_device_fault = 8
```

```
time_fault = 9
```

```
multiple_bond = 10
```

```
class blatann.services.glucose.data_types.GlucoseFeatures(*supported_features)
```

Bases: *Bitfield*

Class which holds the features of the glucose sensor and is reported to over bluetooth. This is the class used for the Feature characteristic.

```
bitfield_width = 16
```

```
bitfield_enum
```

alias of *GlucoseFeatureType*

```
class blatann.services.glucose.data_types.GlucoseSample(glucose_type, sample_location, value,  
                                                         units=GlucoseConcentrationUnits.kg_per_liter)
```

Bases: *BleCompoundDataType*

Holds the info about a glucose sample to be reported through the Glucose Measurement characteristic

**encode()**

**Return type**

*BleDataStream*

**classmethod decode(*stream*)**

**Returns**

The values decoded from the stream

**Return type**

*tuple*

```
class blatann.services.glucose.data_types.GlucoseMeasurement(sequence_number,  
                                                             measurement_time,  
                                                             time_offset_minutes=None,  
                                                             sample=None, sensor_status=None,  
                                                             context=None)
```

Bases: *BleCompoundDataType*

Represents a single measurement taken and can be reported over BLE

**encode()**

**Return type**

*BleDataStream*

**classmethod decode(*stream*)**

**Returns**

The values decoded from the stream

**Return type**

*tuple*

```
class blatann.services.glucose.data_types.CarbsInfo(carbs_grams, carb_type)
```

Bases: *BleCompoundDataType*

Holds information about the carbs consumed

**encode()**

**Return type**

*BleDataStream*

**classmethod decode(*stream*)**

**Returns**

The values decoded from the stream

**Return type**

*tuple*

**class** blatann.services.glucose.data\_types.**ExerciseInfo**(*duration\_seconds, intensity\_percent*)

Bases: *BleCompoundDataType*

Holds information about the exercise performed with the glucose sample

**EXERCISE\_DURATION\_OVERRUN** = 65535

**encode**()

**Return type**

*BleDataStream*

**classmethod decode**(*stream*)

**Returns**

The values decoded from the stream

**Return type**

*tuple*

**class** blatann.services.glucose.data\_types.**MedicationInfo**(*med\_type, med\_value,*  
*med\_units=MedicationUnits.milligrams*)

Bases: *BleCompoundDataType*

Holds information about the medication administered

**encode**()

**Return type**

*BleDataStream*

**classmethod decode**(*stream*)

**Returns**

The values decoded from the stream

**Return type**

*tuple*

**class** blatann.services.glucose.data\_types.**GlucoseContext**(*sequence\_number, carbs=None,*  
*meal\_type=None, tester=None,*  
*health\_status=None, exercise=None,*  
*medication=None, hba1c\_percent=None,*  
*extra\_flags=None*)

Bases: *BleCompoundDataType*

Class which holds the extra glucose context data associated with the glucose measurement

**encode**()

**Return type**

*BleDataStream*

**classmethod decode**(*stream*)

**blatann.services.glucose.database module****class** `blatann.services.glucose.database.IGlucoseDatabase`Bases: `object`

Defines the interface required for the Glucose Service to fetch records and record info

**first\_record()**

Gets the first (oldest) record in the database

**Returns**The first record in the database, or `None` if no records in the database**Return type***GlucoseMeasurement***last\_record()**

Gets the last (newest) record in the database

**Returns**The last record in the database, or `None` if no records in the database**Return type***GlucoseMeasurement***record\_count**(*min\_seq\_num=None, max\_seq\_num=None*)

Gets the number of records between the minimum and maximum sequence numbers provided. The min/max limits are inclusive.

**Parameters**

- **min\_seq\_num** – The minimum sequence number to get. If `None`, no minimum is requested
- **max\_seq\_num** – The maximum sequence number to get. If `None`, no maximum is requested

**Returns**

The number of records that fit the parameters specified

**Return type**`int`**get\_records**(*min\_seq\_num=None, max\_seq\_num=None*)

Gets a list of the records between the minimum sequence and maximum sequence numbers provided. The min/max limits are inclusive.

**Parameters**

- **min\_seq\_num** – The minimum sequence number to get. If `None`, no minimum is requested
- **max\_seq\_num** – The maximum sequence number to get. If `None`, no maximum is requested

**Returns**

The list of glucose measurement records that fit the parameters

**Return type**`list[GlucoseMeasurement]`**delete\_records**(*min\_seq\_num=None, max\_seq\_num=None*)

Deletes the records between the minimum sequence and maximum sequence numbers provided. The min/max limits are inclusive.

**Parameters**

- **min\_seq\_num** – The minimum sequence number to get. If None, no minimum is requested
- **max\_seq\_num** – The maximum sequence number to get. If None, no maximum is requested

**Returns**

The response code to send back for the operation

**Return type**

*RacpResponseCode*

**class** `blatann.services.glucose.database.BasicGlucoseDatabase`(*init\_records=None*)

Bases: *IGlucoseDatabase*

Basic glucose database which simply stores the records in a sorted list, and provides a method for adding new records to the database.

**delete\_records**(*min\_seq\_num=None, max\_seq\_num=None*)

See IGlucoseDatabase

**record\_count**(*min\_seq\_num=None, max\_seq\_num=None*)

See IGlucoseDatabase

**get\_records**(*min\_seq\_num=None, max\_seq\_num=None*)

See IGlucoseDatabase

**first\_record**()

See IGlucoseDatabase

**last\_record**()

See IGlucoseDatabase

**add\_record**(*glucose\_measurement*)

Adds a record to the database. NOTE: the measurement's sequence number must be unique within the database

**Parameters**

**glucose\_measurement** (*GlucoseMeasurement*) – The measurement to add

**blatann.services.glucose.racp module**

**class** `blatann.services.glucose.racp.RacpOpcode`(*value*)

Bases: *IntEnum*

An enumeration.

**report\_stored\_records** = 1

**delete\_stored\_records** = 2

**abort\_operation** = 3

**report\_number\_of\_records** = 4

**number\_of\_records\_response** = 5

**response\_code** = 6

```
class blatann.services.glucose.racp.RacpOperator(value)
    Bases: IntEnum
    An enumeration.
    null = 0
    all_records = 1
    less_than_or_equal_to = 2
    greater_than_or_equal_to = 3
    within_range_inclusive = 4
    first_record = 5
    last_record = 6

class blatann.services.glucose.racp.FilterType(value)
    Bases: IntEnum
    An enumeration.
    sequence_number = 1
    user_facing_time = 2

class blatann.services.glucose.racp.RacpResponseCode(value)
    Bases: IntEnum
    An enumeration.
    success = 1
    not_supported = 2
    invalid_operator = 3
    operator_not_supported = 4
    invalid_operand = 5
    no_records_found = 6
    abort_not_successful = 7
    procedure_not_completed = 8
    operand_not_supported = 9

class blatann.services.glucose.racp.RacpCommand(opcode, operator, filter_type=None,
                                                filter_params=None)
    Bases: BleCompoundDataType
    get_filter_min_max()
    encode()

    Return type
    BleDataStream
```

**classmethod** `decode(stream)`

**Returns**

The values decoded from the stream

**Return type**

`tuple`

**class** `blatann.services.glucose.racp.RacpResponse(request_opcode=None, response_code=None, record_count=None)`

Bases: `BleCompoundDataType`

**encode()**

**Return type**

`BleDataStream`

**classmethod** `decode(stream)`

**Returns**

The values decoded from the stream

**Return type**

`tuple`

## blatann.services.glucose.service module

**class** `blatann.services.glucose.service.GlucoseServer(service, glucose_database, security_level=SecurityLevel.OPEN, include_context_characteristic=True)`

Bases: `object`

**set\_features(features)**

Sets the features for the Glucose Feature characteristic

**Parameters**

**features** (`GlucoseFeatures`) – The supported features of the sensor

**classmethod** `add_to_database(gatts_database, glucose_database, security_level=SecurityLevel.OPEN, include_context_characteristic=True)`

## blatann.services.nordic\_uart package

`blatann.services.nordic_uart.add_nordic_uart_service(gatts_database, max_characteristic_size=None)`

Adds a Nordic UART service to the database

**Parameters**

- **gatts\_database** (`blatann.gatt.gatts.GattsDatabase`) – The database to add the service to
- **max\_characteristic\_size** – The size of the characteristic which determines the read/write chunk size. This should be tuned to the MTU size of the connection

**Returns**

The Nordic Uart Service

**Return type**

`_Server`

`blatann.services.nordic_uart.find_nordic_uart_service(gattc_database)`

Finds a Nordic UART service in the given GATT client database

**Parameters**

**`gattc_database`** (`blatann.gatt.gattc.GattcDatabase`) – the GATT client database to search

**Returns**

The UART service if found, None if not found

**Return type**

`_Client`

## Submodules

`blatann.services.nordic_uart.constants` module

`blatann.services.nordic_uart.service` module

`class blatann.services.nordic_uart.service.NordicUartServer(service, max_characteristic_size=None)`

Bases: `object`

property `on_data_received`

**Return type**

*Event*

property `on_write_complete`

**Return type**

*Event*

property `max_write_length`

`write(data)`

`classmethod add_to_database(gatts_database, max_characteristic_size=None)`

`class blatann.services.nordic_uart.service.NordicUartClient(service)`

Bases: `object`

property `on_data_received`

**Return type**

*Event*

property `on_write_complete`

**Return type**

*Event*



```

property max_write_length
property is_initialized
initialize()
write(data)
classmethod find_in_database(gattc_database)

```

**Return type**  
*NordicUartClient*

## Submodules

### blatann.services.ble\_data\_types module

```
class blatann.services.ble_data_types.BleDataStream(value=b'')
```

```

Bases: object
encode(ble_type, *values)
encode_multiple(*ble_type_value_pairs)
encode_if(conditional, ble_type, *values)
encode_if_multiple(conditional, *ble_type_value_pairs)
decode(ble_type)
decode_if(conditional, ble_type)
decode_multiple(*ble_types)
decode_if_multiple(conditional, *ble_types)
take(num_bytes)
take_all()

```

```
class blatann.services.ble_data_types.BleCompoundDataType
```

```

Bases: object
data_stream_types = []
encode_values(*values)

Return type
BleDataStream

encode()

Return type
BleDataStream

```

**classmethod** `decode(stream)`

**Returns**

The values decoded from the stream

**Return type**

`tuple`

**classmethod** `encoded_size()`

**class** `blatann.services.ble_data_types.BleDataType`

Bases: `object`

**classmethod** `encode(value)`

**classmethod** `decode(stream)`

**classmethod** `encoded_size()`

**class** `blatann.services.ble_data_types.DoubleNibble`

Bases: `BleDataType`

**classmethod** `encode(value)`

**classmethod** `decode(stream)`

**classmethod** `encoded_size()`

**class** `blatann.services.ble_data_types.UnsignedIntegerBase`

Bases: `BleDataType`

**signed** = `False`

**byte\_count** = `1`

**classmethod** `encode(value)`

**classmethod** `decode(stream)`

**classmethod** `encoded_size()`

**class** `blatann.services.ble_data_types.SignedIntegerBase`

Bases: `UnsignedIntegerBase`

**signed** = `True`

**class** `blatann.services.ble_data_types.Int8`

Bases: `SignedIntegerBase`

**byte\_count** = `1`

**class** `blatann.services.ble_data_types.Uint8`

Bases: `UnsignedIntegerBase`

**byte\_count** = `1`

```
class blatann.services.ble_data_types.Int16
    Bases: SignedIntegerBase
    byte_count = 2

class blatann.services.ble_data_types.Uint16
    Bases: UnsignedIntegerBase
    byte_count = 2

class blatann.services.ble_data_types.Uint24
    Bases: UnsignedIntegerBase
    byte_count = 3

class blatann.services.ble_data_types.Uint32
    Bases: UnsignedIntegerBase
    byte_count = 4

class blatann.services.ble_data_types.Int32
    Bases: SignedIntegerBase
    byte_count = 4

class blatann.services.ble_data_types.Uint40
    Bases: UnsignedIntegerBase
    byte_count = 5

class blatann.services.ble_data_types.Uint48
    Bases: UnsignedIntegerBase
    byte_count = 6

class blatann.services.ble_data_types.Uint56
    Bases: UnsignedIntegerBase
    byte_count = 7

class blatann.services.ble_data_types.Uint64
    Bases: UnsignedIntegerBase
    byte_count = 8

class blatann.services.ble_data_types.Int64
    Bases: SignedIntegerBase
    byte_count = 8

class blatann.services.ble_data_types.String
    Bases: BleDataType
    classmethod encode(value)
    classmethod decode(stream)
```

```
class blatann.services.ble_data_types.SFloat
    Bases: BleDataType

    class ReservedMantissaValues
        Bases: object

        POS_INFINITY = 2046

        NEG_INFINITY = 2050

        NAN = 2047

        NRES = 2048

        RESERVED = 2049

        ALL_NAN = [2047, 2048, 2049]

    classmethod encode(value)

    classmethod decode(stream)

    classmethod encoded_size()

class blatann.services.ble_data_types.DateTime(dt)
    Bases: BleCompoundDataType

    data_stream_types = [<class 'blatann.services.ble_data_types.Uint16'>, <class
    'blatann.services.ble_data_types.Uint8'>, <class
    'blatann.services.ble_data_types.Uint8'>, <class
    'blatann.services.ble_data_types.Uint8'>, <class
    'blatann.services.ble_data_types.Uint8'>, <class
    'blatann.services.ble_data_types.Uint8'>]

    encode()

        Return type
            BleDataStream

    classmethod decode(stream)

        Returns
            The values decoded from the stream

        Return type
            tuple

class blatann.services.ble_data_types.DayOfWeek(value)
    Bases: IntEnum

    An enumeration.

    unknown = 0

    monday = 1

    tuesday = 2
```

wednesday = 3

thursday = 4

friday = 5

saturday = 6

sunday = 7

**class** blatann.services.ble\_data\_types.DayDateTime(*dt*)

Bases: *BleCompoundDataType*

**data\_stream\_types** = [<class 'blatann.services.ble\_data\_types.DateTime'>, <class 'blatann.services.ble\_data\_types.Uint8'>]

**encode()**

**Return type**

*BleDataStream*

**classmethod decode**(*stream*)

**Return type**

*datetime.datetime*

**class** blatann.services.ble\_data\_types.Bitfield

Bases: *BleCompoundDataType*

**bitfield\_width** = 8

**bitfield\_enum** = None

**encode()**

**Return type**

*BleDataStream*

**classmethod decode**(*stream*)

**Returns**

The values decoded from the stream

**Return type**

*tuple*

**classmethod from\_integer\_value**(*value*)

**classmethod byte\_count**()

**classmethod encoded\_size**()

**blatann.services.decoded\_event\_dispatcher module**

```
class blatann.services.decoded_event_dispatcher.DecodedReadWriteEventDispatcher(owner,
                                                                              ble_type,
                                                                              event_to_raise,
                                                                              logger=None)
```

Bases: `object`

`decode(data)`

**blatann.utils package**

```
blatann.utils.setup_logger(name=None, level='DEBUG')
```

```
blatann.utils.repr_format(obj, *args, **kwargs)
```

Helper function to format objects into strings in the format of: `ClassName(param1=value1, param2=value2, ...)`

**Parameters**

- **obj** – Object to get the class name from
- **args** – Optional tuples of (param\_name, value) which will ensure ordering during format
- **kwargs** – Other keyword args to populate with

**Returns**

String which represents the object

```
class blatann.utils.Stopwatch
```

Bases: `object`

`start()`

`stop()`

`mark()`

property `is_running`

property `start_time`

property `stop_time`

property `elapsed`

```
class blatann.utils.SynchronousMonotonicCounter(start_value=0)
```

Bases: `object`

Utility class which implements a thread-safe monotonic counter

`next()`

```
blatann.utils.snake_case_to_capitalized_words(string)
```

```
class blatann.utils.IntEnumWithDescription(_, description="")
```

Bases: `int`, `Enum`

An enumeration.

property description

## Submodules

### blatann.utils.queued\_tasks\_manager module

```
class blatann.utils.queued_tasks_manager.QueuedTasksManagerBase(max_processing_items_at_once=1)
    Bases: Generic[T]
    Handles queuing of tasks that can only be done one at a time
    class TaskFailure(reason=None, ignore_stack_trace=False, clear_all=False)
        Bases: object
    clear_all()
```

### blatann.waitables package

## Submodules

### blatann.waitables.connection\_waitable module

```
class blatann.waitables.connection_waitable.ConnectionWaitable(ble_device, current_peer,
                                                                role=BLEGapRoles.periph)
    Bases: Waitable
    wait(timeout=None, exception_on_timeout=True)

    Return type
        blatann.peer.Peer

class blatann.waitables.connection_waitable.ClientConnectionWaitable(ble_device, peer)
    Bases: ConnectionWaitable
    wait(timeout=None, exception_on_timeout=True)

    Return type
        blatann.peer.Client

class blatann.waitables.connection_waitable.PeripheralConnectionWaitable(ble_device, peer)
    Bases: ConnectionWaitable
    wait(timeout=None, exception_on_timeout=True)

    Return type
        blatann.peer.Peripheral

class blatann.waitables.connection_waitable.DisconnectionWaitable(connected_peer)
    Bases: Waitable
```

**blatann.waitables.event\_waitable module****class** blatann.waitables.event\_waitable.**EventWaitable**(*event*)Bases: [Waitable](#), [Generic](#)[[T](#)Sender, [T](#)Event]Waitable implementation which waits on an [Event](#).**wait**(*timeout=None*, *exception\_on\_timeout=True*)

Waits for the asynchronous operation to complete

**Warning:** If this call times out, it cannot be (successfully) called again as it will clean up all event handlers for the waitable. This is done to remove lingering references to the waitable object through event subscriptions

**Parameters**

- **timeout** – How long to wait, or None to wait indefinitely
- **exception\_on\_timeout** – Flag to either throw an exception on timeout, or instead return None object(s)

**Return type**[Tuple](#)[[TypeVar](#)([T](#)Sender), [TypeVar](#)([T](#)Event)]**Returns**

The result of the asynchronous operation

**Raises**[TimeoutError](#)**then**(*callback*)

Registers a function callback that will be called when the asynchronous operation completes

---

**Note:** Only a single callback is supported– subsequent calls to this method will overwrite previous callbacks

---

**Parameters****callback** ([Callable](#)[[[TypeVar](#)([T](#)Sender), [TypeVar](#)([T](#)Event)], [None](#)]) – The function to call when the async operation completes**Returns**

This waitable object

**class** blatann.waitables.event\_waitable.**IdBasedEventWaitable**(*event*, *event\_id*)Bases: [EventWaitable](#), [Generic](#)[[T](#)Sender, [T](#)Event]Extension of [EventWaitable](#) for high-churn events which require IDs to ensure the correct operation is waited upon, such as characteristic read, write and notify operations



**blatann.waitables.scan\_waitable module**

**class** `blatann.waitables.scan_waitable.ScanFinishedWaitable(ble_device)`

Bases: `Waitable`

Waitable that triggers when a scan operation completes. It also provides a mechanism to acquire the received scan reports in real-time

**property** `scan_reports: Iterable[ScanReport]`

Iterable which yields the scan reports in real-time as they're received. The iterable will block until scanning has timed out/finished

**wait**(*timeout=None, exception\_on\_timeout=True*)

Waits for the scanning operation to complete then returns the scan report collection

**Parameters**

- **timeout** (`Optional[float]`) – How long to wait for, in seconds
- **exception\_on\_timeout** (`bool`) – Flag whether or not to throw an exception if the operation timed out. If false and a timeout occurs, will return None

**Return type**

`ScanReportCollection`

**Returns**

The scan report collection

**blatann.waitables.waitable module**

**class** `blatann.waitables.waitable.Waitable(n_args=1)`

Bases: `object`

Base class for an object which can be waited on for an operation to complete. This is a similar concept to `concurrent.futures.Future` where asynchronous operations can block the current thread, or register a handler to be called when it completes.

**wait**(*timeout=None, exception\_on\_timeout=True*)

Waits for the asynchronous operation to complete

**Warning:** If this call times out, it cannot be (successfully) called again as it will clean up all event handlers for the waitable. This is done to remove lingering references to the waitable object through event subscriptions

**Parameters**

- **timeout** (`Optional[float]`) – How long to wait, or None to wait indefinitely
- **exception\_on\_timeout** – Flag to either throw an exception on timeout, or instead return None object(s)

**Returns**

The result of the asynchronous operation

**Raises**

`TimeoutError`

**then**(*callback*)

Registers a function callback that will be called when the asynchronous operation completes

---

**Note:** Only a single callback is supported– subsequent calls to this method will overwrite previous callbacks

---

**Parameters**

**callback** (Callable) – The function to call when the async operation completes

**Returns**

This waitable object

**class** blatann.waitables.waitable.GenericWaitable(*n\_args=1*)

Bases: [Waitable](#)

Simple wrapper of a Waitable object which exposes a **notify** method so external objects can signal/trigger the waitable’s response

**notify**(*\*results*)

**class** blatann.waitables.waitable.EmptyWaitable(*\*args*)

Bases: [Waitable](#)

Waitable class which will immediately return the args provided when waited on or when a callback function is registered

**wait**(*timeout=None, exception\_on\_timeout=True*)

Waits for the asynchronous operation to complete

**Warning:** If this call times out, it cannot be (successfully) called again as it will clean up all event handlers for the waitable. This is done to remove lingering references to the waitable object through event subscriptions

**Parameters**

- **timeout** – How long to wait, or None to wait indefinitely
- **exception\_on\_timeout** – Flag to either throw an exception on timeout, or instead return None object(s)

**Returns**

The result of the asynchronous operation

**Raises**

TimeoutError

**then**(*callback*)

Registers a function callback that will be called when the asynchronous operation completes

---

**Note:** Only a single callback is supported– subsequent calls to this method will overwrite previous callbacks

---

**Parameters**

**callback** – The function to call when the async operation completes

**Returns**

This waitable object

**Submodules****blatann.device module**

```
class blatann.device.BleDevice(comport='COM1', baud=1000000, log_driver_comms=False,  
                               notification_hw_queue_size=16, write_command_hw_queue_size=16,  
                               bond_db_filename='user')
```

Bases: [\*NrfDriverObserver\*](#)

Represents the Bluetooth device itself. Provides the high-level bluetooth APIs (Advertising, Scanning, Connections), configuration, and bond database.

**Parameters**

- **comport** – The port the nRF52 device lives on, e.g. "COM3", "/dev/ttyS0"
- **baud** – The baud rate to use. By default, the connectivity firmware images for v0.3+ use 1M baud.
- **log\_driver\_comms** – debug flag which will enable extra-verbose logging of all communications to the nRF52 hardware
- **notification\_hw\_queue\_size** – Hardware-based queue size to use for notifications. This queue lives within the nRF52 hardware itself and has memory usage implications based on MTU size, etc. This probably won't need to be changed, and from current testing the queue isn't fully exercised
- **write\_command\_hw\_queue\_size** – Hardware-based queue size to use for write commands (write w/o response). Same comments about notification queue apply here.
- **bond\_db\_filename** – Optional filename to use for loading/saving the bonding database. The supported file formats/extensions are: ".pkl" (legacy) and ".json". json is preferred.

Two special values also exist:

- "user" [default] - saves the database within the user's home directory (~/.blatann/bonding\_db.json). This is useful for cases where you may not have write access to the python install location, want to persist the bonding database across virtualenvs, or limit the access to just the logged-in user
- "system" - saves the database within this library's directory structure, wherever it is installed or imported from. Useful if you want the bonding database to be constrained to just that python/virtualenv installation

```
configure(vendor_specific_uuid_count=10, service_changed=False, max_connected_peripherals=1,  
          max_connected_clients=1, max_secured_peripherals=1, attribute_table_size=1408,  
          att_mtu_max_size=247, event_length=6)
```

Configures the BLE Device with the given settings.

---

**Note:** Configuration must be set before opening the device

---

**Parameters**

- **vendor\_specific\_uuid\_count** – The Nordic hardware limits number of 128-bit Base UUIDs that the device can know about. This normally equals the number of custom services that are to be supported, since characteristic UUIDs are usually derived from the service base UUID.
- **service\_changed** – Whether the Service Changed characteristic is exposed in the GAP service
- **max\_connected\_peripherals** – The maximum number of concurrent connections with peripheral devices
- **max\_connected\_clients** – The maximum number of concurrent connections with client devices (NOTE: blatann currently only supports 1)
- **max\_secured\_peripherals** – The maximum number of concurrent peripheral connections that will need security (bonding/pairing) enabled
- **attribute\_table\_size** – The maximum size of the attribute table. Increase this number if there's a lot of services/characteristics in your GATT database.
- **att\_mtu\_max\_size** – The maximum ATT MTU size supported. The default supports an MTU which will fit into a single transmission if Data Length Extensions is set to its max (251)
- **event\_length** – The number of 1.25ms event cycles to dedicate for each connection. The default value (6, =7.5ms) will support the max DLE length of 251 bytes. Minimum value is 2, typical values are 3-8 depending on desired throughput and number of concurrent connections

**open**(*clear\_bonding\_data=False*)

Opens the connection to the BLE device. Must be called prior to performing any BLE operations

**Parameters**

**clear\_bonding\_data** – Flag that the bonding data should be cleared prior to opening the device.

**close**()

Closes the connection to the BLE device. The connection to the device must be opened again to perform BLE operations.

**clear\_bonding\_data**()

Clears out all bonding data from the bond database. Any subsequent connections will require re-pairing.

**property address:** *BLEGapAddr*

The MAC Address of the BLE device

**Getter**

Gets the MAC address of the BLE device

**Setter**

Sets the MAC address for the device to use

---

**Note:** The MAC address cannot be changed while the device is advertising, scanning, or initiating a connection

---

**property database:** [GattsDatabase](#)

**Read Only**

The local database instance that is accessed by connected clients

**property generic\_access\_service:** [GenericAccessService](#)

**Read Only**

The Generic Access service in the local database

**property max\_mtu\_size:** [int](#)

**Read Only**

The maximum allowed ATT MTU size that was configured for the device

---

**Note:** The Max MTU size is set through [configure\(\)](#)

---

**set\_tx\_power**(*tx\_power*)

Sets the radio transmit power. This is used for all connections, advertising, active scanning, etc. Method can be called at any time

Valid transmit power values are -40, -20, -16, -12, -8, -4, 0, 3, and 4 dBm

**Parameters**

**tx\_power** – The transmit power to use, in dBm

**connect**(*peer\_address*, *connection\_params=None*)

Initiates a connection to a peripheral peer with the specified connection parameters, or uses the default connection parameters if not specified. The connection will not be complete until the returned waitable either times out or reports the newly connected peer

**Parameters**

- **peer\_address** ([blatann.gap.gap\\_types.PeerAddress](#)) – The address of the peer to connect to
- **connection\_params** ([blatann.gap.gap\\_types.ConnectionParameters](#)) – Optional connection parameters to use. If not specified, uses the set default

**Return type**

[PeripheralConnectionWaitable](#)

**Returns**

A Waitable which can be used to wait until the connection is successful or times out. Waitable returns a peer.Peripheral object

**set\_default\_peripheral\_connection\_params**(*min\_interval\_ms*, *max\_interval\_ms*, *timeout\_ms*, *slave\_latency=0*)

Sets the default connection parameters for all subsequent connection attempts to peripherals. Refer to the Bluetooth specifications for the valid ranges

**Parameters**

- **min\_interval\_ms** ([float](#)) – The minimum desired connection interval, in milliseconds
- **max\_interval\_ms** ([float](#)) – The maximum desired connection interval, in milliseconds
- **timeout\_ms** ([int](#)) – The connection timeout period, in milliseconds
- **slave\_latency** ([int](#)) – The connection slave latency

**set\_default\_security\_params**(*passcode\_pairing*, *io\_capabilities*, *bond*, *out\_of\_band*,  
*reject\_pairing\_requests=False*, *lesc\_pairing=False*)

Sets the default security parameters for all subsequent connections to peripherals.

#### Parameters

- **passcode\_pairing** (*bool*) – Flag indicating that passcode pairing is required
- **io\_capabilities** (*BLEGapIoCaps*) – The input/output capabilities of this device
- **bond** (*bool*) – Flag indicating that long-term bonding should be performed
- **out\_of\_band** (*bool*) – Flag indicating if out-of-band pairing is supported
- **reject\_pairing\_requests** (*Union[bool, PairingPolicy]*) – Flag indicating that all security requests by the peer should be rejected
- **lesc\_pairing** (*bool*) – Flag indicating that LE Secure Pairing methods are supported

**set\_privacy\_settings**(*enabled*, *resolvable\_address=True*, *update\_rate\_seconds=900*)

Sets the privacy parameters for advertising and connections to the device. When enabled, a random private address will be advertised and updated at the provided interval.

#### Parameters

- **enabled** (*bool*) – True to enable device privacy. Note that only device privacy is supported, network privacy is not
- **resolvable\_address** (*bool*) – True to use a private random resolvable address. If the address is resolvable, bonded peers can use the device's IRK to determine the device's actual public/random address.
- **update\_rate\_seconds** (*int*) – How often the address should be changed/updated, in seconds. Default is 900 (15min)

### blatann.event\_args module

**class** blatann.event\_args.GattOperationCompleteReason(*value*)

Bases: *Enum*

The reason why a GATT operation completed

**SUCCESS** = 0

**QUEUE\_CLEARED** = 1

**CLIENT\_DISCONNECTED** = 2

**SERVER\_DISCONNECTED** = 3

**CLIENT\_UNSUBSCRIBED** = 4

**FAILED** = 5

**TIMED\_OUT** = 6

**class** blatann.event\_args.EventArgs

Bases: *object*

Base Event Arguments class

**class** `blatann.event_args.DisconnectionEventArgs`(*reason*)

Bases: [EventArgs](#)

Event arguments sent when a peer disconnects

**class** `blatann.event_args.MtuSizeUpdatedEventArgs`(*previous\_mtu\_size*, *current\_mtu\_size*)

Bases: [EventArgs](#)

Event arguments for when the effective MTU size on a connection is updated

**class** `blatann.event_args.DataLengthUpdatedEventArgs`(*tx\_bytes*, *rx\_bytes*, *tx\_time\_us*, *rx\_time\_us*)

Bases: [EventArgs](#)

Event arguments for when the Data Length of the link layer has been changed

**class** `blatann.event_args.PhyUpdatedEventArgs`(*status*, *phy\_channel*)

Bases: [EventArgs](#)

Event arguments for when the phy channel is updated

**class** `blatann.event_args.ConnectionParametersUpdatedEventArgs`(*active\_connection\_params*)

Bases: [EventArgs](#)

Event arguments for when connection parameters between peers are updated

**class** `blatann.event_args.SecurityProcess`(*value*)

Bases: [Enum](#)

An enumeration.

**ENCRYPTION** = 0

**PAIRING** = 1

**BONDING** = 1

**class** `blatann.event_args.PairingCompleteEventArgs`(*status*, *security\_level*, *security\_process*)

Bases: [EventArgs](#)

Event arguments when pairing completes, whether it failed or was successful

**class** `blatann.event_args.SecurityLevelChangedEventArgs`(*security\_level*)

Bases: [EventArgs](#)

**class** `blatann.event_args.PasskeyEntryEventArgs`(*key\_type*, *resolve*)

Bases: [EventArgs](#)

Event arguments when a passkey needs to be entered by the user

**resolve**(*passkey=None*)

Submits the passkey entered by the user to the peer

#### Parameters

**passkey** ([Union](#)[[str](#), [int](#), [None](#)]) – The passkey entered by the user. If the key type is passcode, should be a 6-digit string or integer. Use [None](#) or an empty string to cancel.

**class** `blatann.event_args.PasskeyDisplayEventArgs`(*passkey*, *match\_request*, *match\_confirm\_callback*)

Bases: [EventArgs](#)

Event arguments when a passkey needs to be displayed to the user. If *match\_request* is set, the user must confirm that the passkeys match on both devices then send back the confirmation

**match\_confirm**(*keys\_match*)

If key matching was requested, this function responds with whether or not the keys matched correctly :type  
keys\_match: :param keys\_match: True if the keys matched, False if not

**class** blatann.event\_args.**PeripheralSecurityRequestEventArgs**(*bond, mitm, lesc, keypress, is\_bonded\_device, resolver*)

Bases: [EventArgs](#)

Event arguments for when a peripheral requests security to be enabled on the connection. The application must choose how to handle the request: accept, reject, or force re-pairing (if device is bonded).

**class** **Response**(*value*)

Bases: [Enum](#)

An enumeration.

**accept** = 1

**reject** = 2

**force\_repair** = 3

**accept**()

Accepts the security request. If device is already bonded will initiate encryption, otherwise will start the pairing process

**reject**()

Rejects the security request

**force\_repair**()

Accepts the security request and initiates the pairing process, even if the device is already bonded

**class** blatann.event\_args.**PairingRejectedReason**(*value*)

Bases: [Enum](#)

Reason why pairing was rejected

**non\_bonded\_central\_request** = 1

**non\_bonded\_peripheral\_request** = 2

**bonded\_peripheral\_request** = 3

**bonded\_device\_repairing** = 4

**user\_rejected** = 5

**class** blatann.event\_args.**PairingRejectedEventArgs**(*reason*)

Bases: [EventArgs](#)

Event arguments for when a pairing request was rejected locally

**class** blatann.event\_args.**WriteEventArgs**(*value*)

Bases: [EventArgs](#)

Event arguments for when a client has written to a characteristic on the local database



**class** `blatann.event_args.DecodedWriteEventArgs`(*value*, *raw\_value*)

Bases: `EventArgs`, `Generic`[`TDecodedValue`]

Event arguments for when a client has written to a characteristic on the local database and the value has been decoded into a data type

**class** `blatann.event_args.SubscriptionStateChangeEventArgs`(*subscription\_state*)

Bases: `EventArgs`

Event arguments for when a client's subscription state has changed

**class** `blatann.event_args.NotificationCompleteEventArgs`(*notification\_id*, *data*, *reason*)

Bases: `EventArgs`

Event arguments for when a notification has been sent to the client from the notification queue

**Reason**

alias of `GattOperationCompleteReason`

**class** `blatann.event_args.ReadCompleteEventArgs`(*read\_id*, *value*, *status*, *reason*)

Bases: `EventArgs`

Event arguments for when a read has completed of a peripheral's characteristic

**class** `blatann.event_args.WriteCompleteEventArgs`(*write\_id*, *value*, *status*, *reason*)

Bases: `EventArgs`

Event arguments for when a write has completed on a peripheral's characteristic

**class** `blatann.event_args.SubscriptionWriteCompleteEventArgs`(*write\_id*, *value*, *status*, *reason*)

Bases: `EventArgs`

Event arguments for when changing the subscription state of a characteristic completes

**class** `blatann.event_args.NotificationReceivedEventArgs`(*value*, *is\_indication*)

Bases: `EventArgs`

Event Arguments for when a notification or indication is received from the peripheral

**class** `blatann.event_args.DatabaseDiscoveryCompleteEventArgs`(*status*)

Bases: `EventArgs`

Event Arguments for when database discovery completes

**class** `blatann.event_args.DecodedReadCompleteEventArgs`(*read\_id*, *value*, *status*, *reason*,  
*decoded\_stream=None*)

Bases: `ReadCompleteEventArgs`, `Generic`[`TDecodedValue`]

Event Arguments for when a read on a peripheral's characteristic completes and the data stream returned is decoded. If unable to decode the value, the bytes read are still returned

**static** `from_notification_complete_event_args`(*noti\_complete\_event\_args*, *decoded\_stream=None*)

**static** `from_read_complete_event_args`(*read\_complete\_event\_args*, *decoded\_stream=None*)

**blatann.event\_type module****class** blatann.event\_type.**Event**(*name*)Bases: `Generic[TSender, TEvent]`

Represents an event that can have handlers registered and deregistered. All handlers registered to an event should take in two parameters: the event sender and the event arguments.

Those familiar with the C#/.NET event architecture, this should look very similar, though registration is done using the `register()` method instead of `+= event_handler`

**register**(*handler*)

Registers a handler to be called whenever the event is emitted. If the given handler is already registered, function does not register the handler a second time.

This function can be used in a *with* context block which will automatically deregister the handler when the context is exited.

**Example**

```
>>> with device.client.on_connected.register(my_connected_handler):
>>>     # Do something, my_connected_handler will be deregistered upon leaving_
→ this context
```

**Parameters**

**handler** (`Callable[[TypeVar(TSender), TypeVar(TEvent)], None]`) – The handler to register

**Return type**

`EventSubscriptionContext[TypeVar(TSender), TypeVar(TEvent)]`

**Returns**

a context block that can be used to automatically unsubscribe the handler

**deregister**(*handler*)

Deregisters a previously-registered handler so it no longer receives the event. If the given handler is not registered, function does nothing

**Parameters**

**handler** (`Callable[[TypeVar(TSender), TypeVar(TEvent)], None]`) – The handler to deregister

**class** blatann.event\_type.**EventSource**(*name*, *logger=None*)Bases: `Event`

Represents an Event object along with the controls to emit the events and notify handlers. This is done to “hide” the notify method from subscribers.

**property** **has\_handlers**: `bool`

Gets if the event has any handlers subscribed to the event

**clear\_handlers**()

Clears all handlers from the event

**notify**(*sender*, *event\_args=None*)

Notifies all subscribers with the given sender and event arguments

**class** blatann.event\_type.**EventSubscriptionContext**(*event*, *subscriber*)Bases: `Generic[TSender, TEvent]`

## blatann.exceptions module

**exception** `blatann.exceptions.BlatannException`

Bases: `Exception`

**exception** `blatann.exceptions.InvalidStateException`

Bases: `BlatannException`

**exception** `blatann.exceptions.InvalidOperationException`

Bases: `BlatannException`

**exception** `blatann.exceptions.TimeoutError`

Bases: `BlatannException`

**exception** `blatann.exceptions.DecodeError`

Bases: `BlatannException`

## blatann.peer module

**class** `blatann.peer.PeerState(value)`

Bases: `Enum`

Connection state of the peer

**DISCONNECTED** = 0

Peer is disconnected

**CONNECTING** = 1

Peer is in the process of connecting

**CONNECTED** = 2

Peer is connected

**class** `blatann.peer.Peer(ble_device, role, connection_params=ConnectionParams([15-30] ms, timeout: 4000 ms, latency: 0, security_params=SecurityParameters(passcode_pairing=False, io=<BLEGapIoCaps.KEYBOARD_DISPLAY: 4>, bond=False, oob=False, lesc=False), name="", write_no_resp_queue_size=1)`

Bases: `object`

Object that represents a BLE-connected (or disconnected) peer

**BLE\_CONN\_HANDLE\_INVALID** = 65535

Number of bytes that are header/overhead per MTU when sending a notification or indication

**NOTIFICATION\_INDICATION\_OVERHEAD\_BYTES** = 3

**property name:** `str`

The name of the peer, if known. This property is for the user's benefit to name certain connections. The name is also saved in the case that the peer is subsequently bonded to and can be looked up that way in the bond database

---

**Note:** For central peers this name is unknown unless set by the setter. For peripheral peers the name is defaulted to the one found in the advertising payload, if any.

---

**Getter**

Gets the name of the peer

**Setter**

Sets the name of the peer

**property connected:** `bool`

**Read Only**

Gets if this peer is currently connected

**property rssi:** `Optional[int]`

**Read Only**

Gets the RSSI from the latest connection interval, or None if RSSI reporting is not enabled.

---

**Note:** In order for RSSI information to be available, `start_rssi_reporting()` must be called first.

---

**property bytes\_per\_notification:** `int`

**Read Only**

The maximum number of bytes that can be sent in a single notification/indication

**property is\_peripheral:** `bool`

**Read Only**

Gets if this peer is a peripheral (the local device acting as a central/client)

**property is\_client:** `bool`

**Read Only**

Gets if this peer is a Client (the local device acting as a peripheral/server)

**property is\_previously\_bonded:** `bool`

**Read Only**

Gets if the peer has bonding information stored in the bond database (the peer was bonded to in a previous connection)

**property preferred\_connection\_params:** `ConnectionParameters`

**Read Only**

The connection parameters that were negotiated for this peer

**property active\_connection\_params:** `ActiveConnectionParameters`

**Read Only**

The active connection parameters in use with the peer. If the peer is disconnected, this will return the connection parameters last used

**property mtu\_size:** `int`

**Read Only**

The current size of the MTU for the connection to the peer

**property max\_mtu\_size:** `int`

**Read Only**

The maximum allowed MTU size. This is set when initially configuring the BLE Device

**property preferred\_mtu\_size:** `int`

The user-set preferred MTU size. Defaults to the Bluetooth default MTU size (23). This is the value that will be negotiated during an MTU Exchange but is not guaranteed in the case that the peer has a smaller MTU

**Getter**

Gets the preferred MTU size that was configured

**Setter**

Sets the preferred MTU size to use for MTU exchanges

**property preferred\_phy:** `Phy`

The PHY that is preferred for this connection. This value is used for Peer-initiated PHY update procedures and as the default for `update_phy()`.

Default value is `Phy.auto`

**Getter**

Gets the preferred PHY

**Setter**

Sets the preferred PHY

**property phy\_channel:** `Phy`

**Read Only**

The current PHY in use for the connection

**property database:** `GattcDatabase`

**Read Only**

The GATT database of the peer.

---

**Note:** This is not useful until services are discovered first

---

**property on\_connect:** `Event[Peer, None]`

Event generated when the peer connects to the local device

**property on\_disconnect:** `Event[Peer, DisconnectionEventArgs]`

Event generated when the peer disconnects from the local device

**property on\_rssi\_changed:** `Event[Peer, int]`

Event generated when the RSSI has changed for the connection

**property on\_mtu\_exchange\_complete:** `Event[Peer, MtuSizeUpdatedEventArgs]`

Event generated when an MTU exchange completes with the peer

**property on\_mtu\_size\_updated:** `Event[Peer, MtuSizeUpdatedEventArgs]`

Event generated when the effective MTU size has been updated on the connection

**property on\_connection\_parameters\_updated:** `Event[Peer, ConnectionParametersUpdatedEventArgs]`

Event generated when the connection parameters with this peer is updated

**property on\_data\_length\_updated:** `Event[Peer, DataLengthUpdatedEventArgs]`

Event generated when the link layer data length has been updated

**property on\_phy\_updated:** *Event*[*Peer*, *PhyUpdatedEventArgs*]

Event generated when the PHY in use for this peer has been updated

**property on\_database\_discovery\_complete:** *Event*[*Peripheral*, *DatabaseDiscoveryCompleteEventArgs*]

Event that is triggered when database discovery has completed

**disconnect**(*status\_code=BLEHci.remote\_user\_terminated\_connection*)

Disconnects from the peer, giving the optional status code. Returns a waitable that will trigger when the disconnection is complete. If the peer is already disconnected, the waitable will trigger immediately

**Parameters**

**status\_code** – The HCI Status code to send back to the peer

**Return type**

*DisconnectionWaitable*

**Returns**

A waitable that will trigger when the peer is disconnected

**set\_connection\_parameters**(*min\_connection\_interval\_ms*, *max\_connection\_interval\_ms*,  
*connection\_timeout\_ms*, *slave\_latency=0*)

Sets the connection parameters for the peer and starts the connection parameter update process (if connected)

---

**Note:** Connection interval values should be a multiple of 1.25ms since that is the granularity allowed in the Bluetooth specification. Any non-multiples will be rounded down to the nearest 1.25ms. Additionally, the connection timeout has a granularity of 10 milliseconds and will also be rounded as such.

---

**Parameters**

- **min\_connection\_interval\_ms** (*float*) – The minimum acceptable connection interval, in milliseconds
- **max\_connection\_interval\_ms** (*float*) – The maximum acceptable connection interval, in milliseconds
- **connection\_timeout\_ms** (*int*) – The connection timeout, in milliseconds
- **slave\_latency** – The slave latency allowed, which regulates how many connection intervals the peripheral is allowed to skip before responding

**Return type**

*Optional*[*EventWaitable*[*Peer*, *ConnectionParametersUpdatedEventArgs*]]

**Returns**

If the peer is connected, this will return a waitable that will trigger when the update completes with the new connection parameters. If disconnected, returns None

**update\_connection\_parameters**()

Starts the process to re-negotiate the connection parameters using the configured preferred connection parameters

**Return type**

*EventWaitable*[*Peer*, *ConnectionParametersUpdatedEventArgs*]

**Returns**

A waitable that will trigger when the connection parameters are updated

**exchange\_mtu**(*mtu\_size=None*)

Initiates the MTU Exchange sequence with the peer device.

If the MTU size is not provided *preferred\_mtu\_size* value will be used. If an MTU size is provided *preferred\_mtu\_size* will be updated to the given value.

**Parameters**

**mtu\_size** – Optional MTU size to use. If provided, it will also updated the preferred MTU size

**Return type**

*EventWaitable[Peer, MtuSizeUpdatedEventArgs]*

**Returns**

A waitable that will trigger when the MTU exchange completes

**update\_data\_length**(*data\_length=None*)

Starts the process which updates the link layer data length to the optimal value given the MTU. For best results call this method after the MTU is set to the desired size.

**Parameters**

**data\_length** (*Optional[int]*) – Optional value to override the data length to. If not provided, uses the optimal value based on the current MTU

**Return type**

*EventWaitable[Peripheral, DataLengthUpdatedEventArgs]*

**Returns**

A waitable that will trigger when the process finishes

**update\_phy**(*phy=None*)

Performs the PHY update procedure, negotiating a new PHY (1Mbps, 2Mbps, or coded PHY) to use for the connection. Performing this procedure does not guarantee that the PHY will change based on what the peer supports.

**Parameters**

**phy** (*Optional[Phy]*) – Optional PHY to use. If None, uses the *preferred\_phy* attribute. If not None, the preferred PHY is updated to this value.

**Return type**

*EventWaitable[Peer, PhyUpdatedEventArgs]*

**Returns**

An event waitable that triggers when the phy process completes

**discover\_services**()

Starts the database discovery process of the peer. This will discover all services, characteristics, and descriptors on the peer's database.

**Return type**

*EventWaitable[Peer, DatabaseDiscoveryCompleteEventArgs]*

**Returns**

a Waitable that will trigger when service discovery is complete

**start\_rssi\_reporting**(*threshold\_dbm=None, skip\_count=1*)

Starts collecting RSSI readings for the connection

**Parameters**

- **threshold\_dbm** (*Optional[int]*) – Minimum change in dBm before triggering an RSSI changed event. The default value `None` disables the RSSI event (RSSI polled via the *rss\_i* property)
- **skip\_count** – Number of RSSI samples with a change of `threshold_dbm` or more before sending a new RSSI update event. Parameter ignored if `threshold_dbm` is `None`

**Return type***EventWaitable[Peer, int]***Returns**a Waitable that triggers once the first RSSI value is received, if `threshold_dbm` is not `None`**stop\_rssi\_reporting()**Stops collecting RSSI readings. Once stopped, *rss\_i* will return `None`

```
class blatann.peer.Peripheral(ble_device, peer_address, connection_params=ConnectionParams([15-30]
ms, timeout: 4000 ms, latency: 0,
security_params=SecurityParameters(passcode_pairing=False,
io=<BLEGapIoCaps.KEYBOARD_DISPLAY: 4>, bond=False, oob=False,
lesc=False), name="", write_no_resp_queue_size=1)
```

Bases: *Peer*

Object which represents a BLE-connected device that is acting as a peripheral/server (local device is client/central)

**set\_conn\_param\_request\_handler(handler)**

Configures a function callback to handle when a connection parameter request is received from the peripheral and allows the user to decide how to handle the peripheral's requested connection parameters.

The callback is passed in 2 positional parameters: this *Peripheral* object and the desired `ConnectionParameter`s` received in the request. The callback should return the desired connection parameters to use, or `None` to reject the request altogether.

**Parameters**

**handler** (*Callable[[Peripheral, ConnectionParameters], Optional[ConnectionParameters]]*) – The callback to determine which connection parameters to negotiate when an update request is received from the peripheral

**accept\_all\_conn\_param\_requests()**

Sets the connection parameter request handler to a callback that accepts any connection parameter update requests received from the peripheral. This is the same as calling `set_conn_param_request_handler` with a callback that simply returns the connection parameters passed in.

This is the default functionality.

**reject\_conn\_param\_requests()**

Sets the connection parameter request handler to a callback that rejects all connection parameter update requests received from the peripheral. This is same as calling `set_conn_param_request_handler` with a callback that simply returns `None`

```
class blatann.peer.Client(ble_device, connection_params=ConnectionParams([15-30] ms, timeout: 4000
ms, latency: 0, security_params=SecurityParameters(passcode_pairing=False,
io=<BLEGapIoCaps.KEYBOARD_DISPLAY: 4>, bond=False, oob=False,
lesc=False), name="", write_no_resp_queue_size=1)
```

Bases: *Peer*

Object which represents a BLE-connected device that is acting as a client/central (local device is peripheral/server)



**blatann.uuid module**

**class** blatann.uuid.**Uuid**(nrf\_uuid=None, description="")

Bases: `object`

Base class for UUIDs

**property** **descriptive\_string**: `str`

**class** blatann.uuid.**Uuid128**(uuid, description="")

Bases: `Uuid`

Represents a 128-bit UUID

**property** **uuid\_base**: `List[int]`

**Read Only**

The base of the 128-bit UUID which can be used to create other UUIDs with the same base

**property** **uuid16**: `int`

**Read Only**

The 16-bit representation of the 128-bit UUID

**new\_uuid\_from\_base**(uuid16)

Creates a new 128-bit UUID with the same base as this UUID, replacing out the 16-bit individual identifier with the value provided

**Parameters**

**uuid16** (`Union[str, int, Uuid16]`) – The new 16-bit UUID to append to the base. Should either be a 16-bit integer, a hex string of the value (without the leading ‘0x’), or a `Uuid16` object

**Return type**

`Uuid128`

**Returns**

The newly created UUID

**classmethod** **combine\_with\_base**(uuid16, uuid128\_base)

Combines a 16-bit UUID with a 128-bit UUID base and returns the new UUID

**Parameters**

- **uuid16** (`Union[str, int, Uuid16]`) – The 16-bit UUID to use. See `new_uuid_from_base` for format.
- **uuid128\_base** (`Union[str, bytes, List[int]]`) – The 128-bit base UUID to use. See `__init__` for format.

**Return type**

`Uuid128`

**Returns**

The created UUID

**class** blatann.uuid.**Uuid16**(uuid, description="")

Bases: `Uuid`

Represents a 16-bit “short form” UUID

`blatann.uuid.generate_random_uuid16()`

Generates a random 16-bit UUID

**Return type**

*Uuid16*

**Returns**

The generated 16-bit UUID

`blatann.uuid.generate_random_uuid128()`

Generates a random 128-bit UUID

**Return type**

*Uuid128*

**Returns**

The generated 128-bit UUID

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### b

- blatann, 23
- blatann.bt\_sig, 23
- blatann.bt\_sig.assigned\_numbers, 23
- blatann.bt\_sig.uuids, 31
- blatann.device, 151
- blatann.event\_args, 154
- blatann.event\_type, 158
- blatann.examples, 49
- blatann.examples.broadcaster, 49
- blatann.examples.central\_uart\_service, 49
- blatann.examples.central, 50
- blatann.examples.central\_battery\_service, 51
- blatann.examples.central\_descriptors, 51
- blatann.examples.central\_device\_info\_service, 51
- blatann.examples.central\_event\_driven, 51
- blatann.examples.constants, 52
- blatann.examples.example\_utils, 52
- blatann.examples.peripheral, 52
- blatann.examples.peripheral\_battery\_service, 54
- blatann.examples.peripheral\_current\_time\_service, 55
- blatann.examples.peripheral\_descriptors, 56
- blatann.examples.peripheral\_device\_info\_service, 56
- blatann.examples.peripheral\_glucose\_service, 57
- blatann.examples.peripheral\_rssi, 58
- blatann.examples.peripheral\_uart\_service, 58
- blatann.examples.scanner, 59
- blatann.exceptions, 159
- blatann.gap, 59
- blatann.gap.advertise\_data, 59
- blatann.gap.advertising, 64
- blatann.gap.bond\_db, 66
- blatann.gap.default\_bond\_db, 67
- blatann.gap.gap\_types, 69
- blatann.gap.generic\_access\_service, 70
- blatann.gap.scanning, 71
- blatann.gap.smp, 72
- blatann.gap.smp\_crypto, 75
- blatann.gatt, 76
- blatann.gatt.gattc, 78
- blatann.gatt.gattc\_attribute, 83
- blatann.gatt.gatts, 83
- blatann.gatt.gatts\_attribute, 88
- blatann.gatt.managers, 90
- blatann.gatt.reader, 90
- blatann.gatt.service\_discovery, 91
- blatann.gatt.writer, 91
- blatann.nrf, 92
- blatann.nrf.nrf\_dll\_load, 116
- blatann.nrf.nrf\_driver, 116
- blatann.nrf.nrf\_driver\_types, 119
- blatann.nrf.nrf\_events, 92
- blatann.nrf.nrf\_events.gap\_events, 92
- blatann.nrf.nrf\_events.gatt\_events, 94
- blatann.nrf.nrf\_events.generic\_events, 96
- blatann.nrf.nrf\_events.smp\_events, 97
- blatann.nrf.nrf\_types, 98
- blatann.nrf.nrf\_types.config, 98
- blatann.nrf.nrf\_types.enums, 100
- blatann.nrf.nrf\_types.gap, 108
- blatann.nrf.nrf\_types.gatt, 110
- blatann.nrf.nrf\_types.generic, 113
- blatann.nrf.nrf\_types.smp, 114
- blatann.peer, 159
- blatann.services, 120
- blatann.services.battery, 120
- blatann.services.battery.constants, 121
- blatann.services.battery.data\_types, 121
- blatann.services.battery.service, 121
- blatann.services.ble\_data\_types, 141
- blatann.services.current\_time, 122
- blatann.services.current\_time.constants, 123
- blatann.services.current\_time.data\_types, 123
- blatann.services.current\_time.service, 125
- blatann.services.decoded\_event\_dispatcher, 146
- blatann.services.device\_info, 127
- blatann.services.device\_info.constants, 128
- blatann.services.device\_info.data\_types, 128

`blatann.services.device_info.service`, 129  
`blatann.services.glucose`, 130  
`blatann.services.glucose.constants`, 130  
`blatann.services.glucose.data_types`, 130  
`blatann.services.glucose.database`, 136  
`blatann.services.glucose.racp`, 137  
`blatann.services.glucose.service`, 139  
`blatann.services.nordic_uart`, 139  
`blatann.services.nordic_uart.constants`, 140  
`blatann.services.nordic_uart.service`, 140  
`blatann.utils`, 146  
`blatann.utils.queued_tasks_manager`, 147  
`blatann.uuid`, 165  
`blatann.waitables`, 147  
`blatann.waitables.connection_waitable`, 147  
`blatann.waitables.event_waitable`, 148  
`blatann.waitables.scan_waitable`, 149  
`blatann.waitables.waitable`, 149

## A

- `abort_not_successful` (blatann.services.glucose.racp.RacpResponseCode attribute), 138
- `abort_operation` (blatann.services.glucose.racp.RacpOpcode attribute), 137
- `absorbed_dose_gray` (blatann.bt\_sig.assigned\_numbers.Units attribute), 25
- `absorbed_dose_rate_gray_per_second` (blatann.bt\_sig.assigned\_numbers.Units attribute), 25
- `acceleration_metres_per_second_squared` (blatann.bt\_sig.assigned\_numbers.Units attribute), 25
- `accept` (blatann.event\_args.PeripheralSecurityRequestEventArgs.Response attribute), 156
- `accept()` (blatann.event\_args.PeripheralSecurityRequestEventArgs method), 156
- `accept_all_conn_param_requests()` (blatann.peer.Peripheral method), 164
- `active_connection_params` (blatann.peer.Peer property), 160
- `active_preset_index` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 34
- `ActiveConnectionParameters` (class in blatann.gap.gap\_types), 70
- `activity_current_session` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 34
- `activity_goal` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 34
- `activity_referred_to_a_radionuclide_becquerel` (blatann.bt\_sig.assigned\_numbers.Units attribute), 25
- `add()` (blatann.gap.bond\_db.BondDatabase method), 66
- `add()` (blatann.gap.default\_bond\_db.DefaultBondDatabase method), 69
- `add_battery_service()` (in module blatann.services.battery), 120
- `add_characteristic()` (blatann.gatt.gatts.GattsService method), 87
- `add_constant_value_descriptor()` (blatann.gatt.gatts.GattsCharacteristic method), 85
- `add_current_time_service()` (in module blatann.services.current\_time), 122
- `add_descriptor()` (blatann.gatt.gatts.GattsCharacteristic method), 85
- `add_device_info_service()` (in module blatann.services.device\_info), 127
- `add_fake_glucose_readings()` (in module blatann.examples.peripheral\_glucose\_service), 57
- `add_glucose_service()` (in module blatann.services.glucose), 130
- `add_nordic_uart_service()` (in module blatann.services.nordic\_uart), 139
- `add_record()` (blatann.services.glucose.database.BasicGlucoseDatabase method), 137
- `add_service()` (blatann.gatt.gatts.GattsDatabase method), 88
- `add_to_database()` (blatann.services.battery.service.BatteryServer class method), 121
- `add_to_database()` (blatann.services.current\_time.service.CurrentTimeServer class method), 126
- `add_to_database()` (blatann.services.device\_info.service.DisServer class method), 129
- `add_to_database()` (blatann.services.glucose.service.GlucoseServer class method), 139
- `add_to_database()` (blatann.services.nordic\_uart.service.NordicUartServer class method), 140
- `address` (blatann.device.BleDevice property), 152
- `AdjustmentReason` (class in blatann.services.current\_time.data\_types), 124
- `AdjustmentReasonType` (class in blatann.services.current\_time.data\_types), 124

<i>tann.services.current_time.data_types</i> ), 123	31	
<code>adv_constant_tone_interval</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	<code>alert_category_id</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	
<code>adv_constant_tone_min_length</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	<code>alert_category_id_bit_mask</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	
<code>adv_constant_tone_min_tx_count</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	<code>alert_level</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	
<code>adv_constant_tone_phy</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	<code>alert_notification</code> ( <i>blatann.bt_sig.uuids.ServiceUuid attribute</i> ), 32	
<code>adv_constant_tone_tx_duration</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	<code>alert_notification_control_point</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	
<code>adv_params_setup()</code> ( <i>blatann.nrf.nrf_driver.NrfDriver method</i> ), 117	<code>alert_status</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	
<code>ADVERTISE_FOREVER</code> ( <i>blatann.gap.advertising.Advertiser attribute</i> ), 64	<code>ALL_NAN</code> ( <i>blatann.services.ble_data_types.SFloat.ReservedMantissaValues attribute</i> ), 144	
<code>Advertiser</code> (class in <i>blatann.gap.advertising</i> ), 64	<code>all_records</code> ( <i>blatann.services.glucose.racp.RacpOperator attribute</i> ), 138	
<code>advertising</code> ( <i>blatann.nrf.nrf_types.enums.BLEGapTimeoutSrc attribute</i> ), 104	<code>all_scan_reports</code> ( <i>blatann.gap.advertise_data.ScanReportCollection property</i> ), 63	
<code>advertising_interval</code> ( <i>blatann.gap.advertise_data.AdvertisingData.Types attribute</i> ), 61	<code>allow_all</code> ( <i>blatann.gap.smp.PairingPolicy attribute</i> ), 72	
<code>advertising_interval</code> ( <i>blatann.nrf.nrf_types.gap.BLEAdvData.Types attribute</i> ), 109	<code>alternate_test_site</code> ( <i>blatann.services.glucose.data_types.SampleLocation attribute</i> ), 131	
<code>advertising_peers_found</code> ( <i>blatann.gap.advertise_data.ScanReportCollection property</i> ), 63	<code>altitude</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	
<code>AdvertisingData</code> (class in <i>blatann.gap.advertise_data</i> ), 59	<code>ammonia_concentration</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	
<code>AdvertisingData.Types</code> (class in <i>blatann.gap.advertise_data</i> ), 60	<code>amount_concentration_mole_per_cubic_metre</code> ( <i>blatann.bt_sig.assigned_numbers.Units attribute</i> ), 25	
<code>AdvertisingFlags</code> (class in <i>blatann.gap.advertise_data</i> ), 59	<code>amount_of_substance_mole</code> ( <i>blatann.bt_sig.assigned_numbers.Units attribute</i> ), 25	
<code>aerobic_heart_rate_lower_limit</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	<code>anaerobic_heart_rate_lower_limit</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	
<code>aerobic_heart_rate_upper_limit</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	<code>anaerobic_heart_rate_upper_limit</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	
<code>aerobic_threshold</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	<code>anaerobic_threshold</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	
<code>age</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	<code>analog</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	
<code>aggregate</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 34	<code>analog_output</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid attribute</i> ), 35	
<code>aggregate_format</code> ( <i>blatann.bt_sig.uuids.DescriptorUuid attribute</i> ),	<code>angular_acceleration_radian_per_second_squared</code> ( <i>blatann.bt_sig.assigned_numbers.Units attribute</i> ),	



tribute), 25

angular\_velocity\_radian\_per\_second (blatann.bt\_sig.assigned\_numbers.Units attribute), 25

angular\_velocity\_revolution\_per\_minute (blatann.bt\_sig.assigned\_numbers.Units attribute), 25

anonymous (blatann.nrf.nrf\_types.gap.BLEGapAddrTypes attribute), 108

app\_begin (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 107

app\_end (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 107

apparent\_wind\_direction (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35

apparent\_wind\_speed (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35

appearance (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35

appearance (blatann.gap.advertise\_data.AdvertisingData.Types attribute), 60

appearance (blatann.gap.generic\_access\_service.GenericAccessService attribute), 70

appearance (blatann.nrf.nrf\_types.gap.BLEAdvData.Types attribute), 109

Appearance (class in blatann.bt\_sig.assigned\_numbers), 29

area\_barn (blatann.bt\_sig.assigned\_numbers.Units attribute), 25

area\_hectare (blatann.bt\_sig.assigned\_numbers.Units attribute), 25

area\_square\_metres (blatann.bt\_sig.assigned\_numbers.Units attribute), 25

arterial\_plasma (blatann.services.glucose.data\_types.GlucoseType attribute), 130

arterial\_whole\_blood (blatann.services.glucose.data\_types.GlucoseType attribute), 130

as\_array() (blatann.nrf.nrf\_types.generic.BLEUUID method), 113

as\_bytes() (blatann.bt\_sig.assigned\_numbers.Appearance method), 31

ase\_control\_point (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35

atomic\_clock (blatann.services.current\_time.data\_types.TimeSource attribute), 123

ATT\_MTU\_DEFAULT (blatann.nrf.nrf\_driver.NrfDriver attribute), 117

attr\_info128\_array\_to\_list() (in module blatann.nrf.nrf\_driver\_types), 119

attr\_info16\_array\_to\_list() (in module blatann.nrf.nrf\_driver\_types), 119

attr\_info\_array\_to\_list() (in module blatann.nrf.nrf\_driver\_types), 119

Attribute (class in blatann.gatt), 77

attribute\_not\_found (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106

attribute\_not\_long (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106

attributes (blatann.gatt.gattc.GattcCharacteristic property), 79

attributes (blatann.gatt.gatts.GattsCharacteristic property), 86

audio\_input\_control (blatann.bt\_sig.uuids.ServiceUuid attribute), 32

audio\_input\_control\_point (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35

audio\_input\_description (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35

audio\_input\_state (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35

audio\_input\_status (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35

audio\_input\_type (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35

audio\_location (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35

audio\_output\_description (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35

audio\_stream\_control (blatann.bt\_sig.uuids.ServiceUuid attribute), 32

auth\_req (blatann.nrf.nrf\_types.enums.BLEGapSecStatus attribute), 105

authentication\_failure (blatann.nrf.nrf\_types.enums.BLEHci attribute), 101

auto (blatann.gap.gap\_types.Phy attribute), 69

auto\_source (blatann.nrf.nrf\_types.enums.BLEGapPhy attribute), 104

auto\_restart (blatann.gap.advertising.Advertiser property), 64

automation\_io (blatann.bt\_sig.uuids.ServiceUuid attribute), 32

- tribute), 32
- auxiliary (blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor attribute), 24
- available\_audio\_contexts (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- average\_current (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- average\_voltage (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- ## B
- back (blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor attribute), 24
- backup (blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor attribute), 24
- barcode\_scanner (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 29
- barometric\_pressure\_trend (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- basic\_audio\_announcement (blatann.bt\_sig.uuids.ServiceUuid attribute), 32
- BasicGlucoseDatabase (class in blatann.services.glucose.database), 137
- battery\_level (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- battery\_level (blatann.nrf.nrf\_types.generic.BLEUUID.Standard attribute), 113
- battery\_level\_state (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- battery\_low (blatann.services.glucose.data\_types.SensorStatusType attribute), 132
- battery\_power\_state (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- battery\_service (blatann.bt\_sig.uuids.ServiceUuid attribute), 32
- BatteryClient (class in blatann.services.battery.service), 121
- BatteryLevel (class in blatann.services.battery.data\_types), 121
- BatteryServer (class in blatann.services.battery.service), 121
- bearer\_list\_current\_calls (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- bearer\_provider\_name (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- bearer\_signal\_strength (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- bearer\_signal\_strength\_reporting\_interval (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- bearer\_technology (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- bearer\_uci (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- bearer\_uri\_schemes\_supported\_list (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35
- bedtime (blatann.services.glucose.data\_types.MealType attribute), 131
- binary\_sensor (blatann.bt\_sig.uuids.ServiceUuid attribute), 32
- Bitfield (class in blatann.services.ble\_data\_types), 145
- bitfield\_enum (blatann.services.ble\_data\_types.Bitfield attribute), 145
- bitfield\_enum (blatann.services.current\_time.data\_types.AdjustmentReason attribute), 124
- bitfield\_enum (blatann.services.glucose.data\_types.GlucoseFeatures attribute), 133
- bitfield\_enum (blatann.services.glucose.data\_types.SensorStatus attribute), 133
- bitfield\_width (blatann.services.ble\_data\_types.Bitfield attribute), 145
- bitfield\_width (blatann.services.current\_time.data\_types.AdjustmentReason attribute), 124
- bitfield\_width (blatann.services.glucose.data\_types.GlucoseFeatures attribute), 133
- bitfield\_width (blatann.services.glucose.data\_types.SensorStatus attribute), 133
- blatann
- module, 23
- blatann.bt\_sig
- module, 23
- blatann.bt\_sig.assigned\_numbers
- module, 23
- blatann.bt\_sig.uuids
- module, 31
- blatann.device
- module, 151
- blatann.event\_args
- module, 154
- blatann.event\_type
- module, 158

blatann.examples	blatann.gap.scanning
module, 49	module, 71
blatann.examples.broadcaster	blatann.gap.smp
module, 49	module, 72
blatann.examples.central_uart_service	blatann.gap.smp_crypto
module, 49	module, 75
blatann.examples.central	blatann.gatt
module, 50	module, 76
blatann.examples.central_battery_service	blatann.gatt.gattc
module, 51	module, 78
blatann.examples.central_descriptors	blatann.gatt.gattc_attribute
module, 51	module, 83
blatann.examples.central_device_info_service	blatann.gatt.gatts
module, 51	module, 83
blatann.examples.central_event_driven	blatann.gatt.gatts_attribute
module, 51	module, 88
blatann.examples.constants	blatann.gatt.managers
module, 52	module, 90
blatann.examples.example_utils	blatann.gatt.reader
module, 52	module, 90
blatann.examples.peripheral	blatann.gatt.service_discovery
module, 52	module, 91
blatann.examples.peripheral_battery_service	blatann.gatt.writer
module, 54	module, 91
blatann.examples.peripheral_current_time_service	blatann.nrf
module, 55	module, 92
blatann.examples.peripheral_descriptors	blatann.nrf.nrf_dll_load
module, 56	module, 116
blatann.examples.peripheral_device_info_service	blatann.nrf.nrf_driver
module, 56	module, 116
blatann.examples.peripheral_glucose_service	blatann.nrf.nrf_driver_types
module, 57	module, 119
blatann.examples.peripheral_rssi	blatann.nrf.nrf_events
module, 58	module, 92
blatann.examples.peripheral_uart_service	blatann.nrf.nrf_events.gap_events
module, 58	module, 92
blatann.examples.scanner	blatann.nrf.nrf_events.gatt_events
module, 59	module, 94
blatann.exceptions	blatann.nrf.nrf_events.generic_events
module, 159	module, 96
blatann.gap	blatann.nrf.nrf_events.smp_events
module, 59	module, 97
blatann.gap.advertise_data	blatann.nrf.nrf_types
module, 59	module, 98
blatann.gap.advertising	blatann.nrf.nrf_types.config
module, 64	module, 98
blatann.gap.bond_db	blatann.nrf.nrf_types.enums
module, 66	module, 100
blatann.gap.default_bond_db	blatann.nrf.nrf_types.gap
module, 67	module, 108
blatann.gap.gap_types	blatann.nrf.nrf_types.gatt
module, 69	module, 110
blatann.gap.generic_access_service	blatann.nrf.nrf_types.generic
module, 70	module, 113

`blatann.nrf.nrf_types.smp`  
    module, 114

`blatann.peer`  
    module, 159

`blatann.services`  
    module, 120

`blatann.services.battery`  
    module, 120

`blatann.services.battery.constants`  
    module, 121

`blatann.services.battery.data_types`  
    module, 121

`blatann.services.battery.service`  
    module, 121

`blatann.services.ble_data_types`  
    module, 141

`blatann.services.current_time`  
    module, 122

`blatann.services.current_time.constants`  
    module, 123

`blatann.services.current_time.data_types`  
    module, 123

`blatann.services.current_time.service`  
    module, 125

`blatann.services.decoded_event_dispatcher`  
    module, 146

`blatann.services.device_info`  
    module, 127

`blatann.services.device_info.constants`  
    module, 128

`blatann.services.device_info.data_types`  
    module, 128

`blatann.services.device_info.service`  
    module, 129

`blatann.services.glucose`  
    module, 130

`blatann.services.glucose.constants`  
    module, 130

`blatann.services.glucose.data_types`  
    module, 130

`blatann.services.glucose.database`  
    module, 136

`blatann.services.glucose.racp`  
    module, 137

`blatann.services.glucose.service`  
    module, 139

`blatann.services.nordic_uart`  
    module, 139

`blatann.services.nordic_uart.constants`  
    module, 140

`blatann.services.nordic_uart.service`  
    module, 140

`blatann.utils`  
    module, 146

`blatann.utils.queued_tasks_manager`  
    module, 147

`blatann.uuid`  
    module, 165

`blatann.waitables`  
    module, 147

`blatann.waitables.connection_waitable`  
    module, 147

`blatann.waitables.event_waitable`  
    module, 148

`blatann.waitables.scan_waitable`  
    module, 149

`blatann.waitables.waitable`  
    module, 149

`BlatannException`, 159

`ble_ah()` (in module `blatann.gap.smp_crypto`), 76

`ble_blocked_by_other_links` (*blatann.nrf.nrf\_types.enums.NrfError* attribute), 102

`ble_conn_configure()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 117

`BLE_CONN_HANDLE_INVALID` (*blatann.peer.Peer* attribute), 159

`ble_enable()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 117

`ble_enable_params_setup()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 117

`ble_evt_handler()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 119

`ble_gap_addr_get()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 117

`ble_gap_addr_set()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 117

`ble_gap_adv_data_set()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 118

`ble_gap_adv_start()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 117

`ble_gap_adv_stop()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 118

`ble_gap_appearance_set()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 117

`ble_gap_auth_key_reply()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 118

`ble_gap_authenticate()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 118

`ble_gap_conn_param_update()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 117

`ble_gap_connect()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 118

`ble_gap_data_length_update()` (*blatann.nrf.nrf\_driver.NrfDriver* method), 118

`ble_gap_device_identities_duplicate` (*blatann.nrf.nrf\_types.enums.NrfError* attribute), 102

<code>ble_gap_device_identities_in_use</code>	( <i>blatann.nrf.nrf_types.enums.NrfError</i> attribute), 102	<code>ble_gattc_char_disc()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_device_name_set()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 117	<code>ble_gattc_desc_disc()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_disconnect()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118	<code>ble_gattc_exchange_mtu_req()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 119
<code>ble_gap_discoverable_with_whitelist</code>	( <i>blatann.nrf.nrf_types.enums.NrfError</i> attribute), 102	<code>ble_gattc_hv_confirm()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 119
<code>ble_gap_encrypt()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118	<code>ble_gattc_prim_srvc_disc()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_invalid_ble_addr</code>	( <i>blatann.nrf.nrf_types.enums.NrfError</i> attribute), 102	<code>ble_gattc_proc_not_permitted</code>	( <i>blatann.nrf.nrf_types.enums.NrfError</i> attribute), 102
<code>ble_gap_lesc_dhkey_reply()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118	<code>ble_gattc_read()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 119
<code>ble_gap_phy_update()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118	<code>ble_gattc_write()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_ppcp_set()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 117	<code>ble_gatts_characteristic_add()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_privacy_set()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 117	<code>ble_gatts_descriptor_add()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_rssi_get()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118	<code>ble_gatts_exchange_mtu_reply()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_rssi_start()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118	<code>ble_gatts_hvx()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_rssi_stop()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118	<code>ble_gatts_invalid_attr_type</code>	( <i>blatann.nrf.nrf_types.enums.NrfError</i> attribute), 103
<code>ble_gap_scan_start()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118	<code>ble_gatts_rw_authorize_reply()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_scan_stop()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118	<code>ble_gatts_service_add()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_sec_info_reply()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118	<code>ble_gatts_service_changed()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_sec_params_reply()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118	<code>ble_gatts_sys_attr_missing</code>	( <i>blatann.nrf.nrf_types.enums.NrfError</i> attribute), 103
<code>ble_gap_tx_power_set()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 117	<code>ble_gatts_sys_attr_set()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_uuid_list_mismatch</code>	( <i>blatann.nrf.nrf_types.enums.NrfError</i> attribute), 102	<code>ble_gatts_value_get()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>ble_gap_whitelist_in_use</code>	( <i>blatann.nrf.nrf_types.enums.NrfError</i> attribute), 102	<code>ble_gatts_value_set()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 118
<code>BLE_GATT_HANDLE_INVALID</code>	(in module <i>blatann.nrf.nrf_types.gatt</i> ), 110	<code>ble_invalid_adv_handle</code>	( <i>blatann.nrf.nrf_types.enums.NrfError</i> attribute), 102
<code>ble_gattc_attr_info128_array_to_list()</code>	(in module <i>blatann.nrf.nrf_driver_types</i> ), 119	<code>ble_invalid_attr_handle</code>	( <i>blatann.nrf.nrf_types.enums.NrfError</i> attribute), 102
<code>ble_gattc_attr_info16_array_to_list()</code>	(in module <i>blatann.nrf.nrf_driver_types</i> ), 119	<code>ble_invalid_conn_handle</code>	( <i>blatann.nrf.nrf_types.enums.NrfError</i> attribute), 102
<code>ble_gattc_attr_info_disc()</code>	( <i>blatann.nrf.nrf_driver.NrfDriver</i> method), 119	<code>ble_invalid_role</code>	( <i>blatann.nrf.nrf_driver_types</i> ), 119
<code>ble_gattc_char_array_to_list()</code>	(in module <i>blatann.nrf.nrf_driver_types</i> ), 119		



`tann.nrf.nrf_types.enums.NrfError` attribute), 102  
`ble_not_enabled` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 102  
`ble_opt_set()` (`blatann.nrf.nrf_driver.NrfDriver` method), 117  
`ble_user_mem_reply()` (`blatann.nrf.nrf_driver.NrfDriver` method), 117  
`BLE_UUID_TYPE_BLE` (`blatann.nrf.nrf_types.generic.BLEUUIDBase` attribute), 113  
`ble_vs_uuid_add()` (`blatann.nrf.nrf_driver.NrfDriver` method), 117  
`BLEAdvData` (class in `blatann.nrf.nrf_types.gap`), 109  
`BLEAdvData.Types` (class in `blatann.nrf.nrf_types.gap`), 109  
`BleCompoundDataType` (class in `blatann.services.ble_data_types`), 141  
`BleConnConfig` (class in `blatann.nrf.nrf_types.config`), 100  
`BleDataStream` (class in `blatann.services.ble_data_types`), 141  
`BleDataType` (class in `blatann.services.ble_data_types`), 142  
`BleDevice` (class in `blatann.device`), 151  
`BleEnableConfig` (class in `blatann.nrf.nrf_types.config`), 100  
`BleEnableOpt` (class in `blatann.nrf.nrf_types.config`), 98  
`BLEEvent` (class in `blatann.nrf.nrf_events.generic_events`), 96  
`BLEGapAddr` (class in `blatann.nrf.nrf_types.gap`), 108  
`BLEGapAddrTypes` (class in `blatann.nrf.nrf_types.gap`), 108  
`BLEGapAdvParams` (class in `blatann.nrf.nrf_types.gap`), 108  
`BLEGapAdvType` (class in `blatann.nrf.nrf_types.enums`), 103  
`BLEGapAuthKeyType` (class in `blatann.nrf.nrf_types.enums`), 105  
`BLEGapConnParams` (class in `blatann.nrf.nrf_types.gap`), 108  
`BLEGapDataLengthParams` (class in `blatann.nrf.nrf_types.gap`), 110  
`BLEGapDhKey` (class in `blatann.nrf.nrf_types.smp`), 116  
`BLEGapEncryptInfo` (class in `blatann.nrf.nrf_types.smp`), 115  
`BLEGapEncryptKey` (class in `blatann.nrf.nrf_types.smp`), 115  
`BLEGapIdKey` (class in `blatann.nrf.nrf_types.smp`), 115  
`BLEGapIoCaps` (class in `blatann.nrf.nrf_types.enums`), 104  
`BLEGapMasterId` (class in `blatann.nrf.nrf_types.smp`), 115  
`BLEGapPhy` (class in `blatann.nrf.nrf_types.enums`), 104  
`BLEGapPhys` (class in `blatann.nrf.nrf_types.gap`), 110  
`BLEGapPrivacyParams` (class in `blatann.nrf.nrf_types.gap`), 110  
`BLEGapPublicKey` (class in `blatann.nrf.nrf_types.smp`), 115  
`BLEGapRoles` (class in `blatann.nrf.nrf_types.enums`), 104  
`BLEGapScanParams` (class in `blatann.nrf.nrf_types.gap`), 108  
`BLEGapSecKeyDist` (class in `blatann.nrf.nrf_types.smp`), 114  
`BLEGapSecKeys` (class in `blatann.nrf.nrf_types.smp`), 116  
`BLEGapSecKeyset` (class in `blatann.nrf.nrf_types.smp`), 116  
`BLEGapSecLevels` (class in `blatann.nrf.nrf_types.smp`), 114  
`BLEGapSecMode` (class in `blatann.nrf.nrf_types.smp`), 114  
`BLEGapSecModeType` (class in `blatann.nrf.nrf_types.smp`), 114  
`BLEGapSecParams` (class in `blatann.nrf.nrf_types.smp`), 114  
`BLEGapSecStatus` (class in `blatann.nrf.nrf_types.enums`), 105  
`BLEGapSignKey` (class in `blatann.nrf.nrf_types.smp`), 116  
`BLEGapTimeoutSrc` (class in `blatann.nrf.nrf_types.enums`), 104  
`BLEGattAttrInfo128` (class in `blatann.nrf.nrf_types.gatt`), 112  
`BLEGattAttrInfo16` (class in `blatann.nrf.nrf_types.gatt`), 111  
`BLEGattDescriptor` (class in `blatann.nrf.nrf_types.gatt`), 111  
`BLEGattCharacteristic` (class in `blatann.nrf.nrf_types.gatt`), 111  
`BLEGattCharacteristicProperties` (class in `blatann.nrf.nrf_types.gatt`), 110  
`BLEGattWriteParams` (class in `blatann.nrf.nrf_types.gatt`), 111  
`BleGattEnableParams` (class in `blatann.nrf.nrf_types.gatt`), 110  
`BLEGattExecWriteFlag` (class in `blatann.nrf.nrf_types.enums`), 107  
`BLEGattExtendedCharacteristicProperties` (class in `blatann.nrf.nrf_types.gatt`), 111  
`BleGattHandle` (class in `blatann.nrf.nrf_types.gatt`), 111  
`BLEGattHVXType` (class in `blatann.nrf.nrf_types.enums`), 106  
`BLEGattsAttribute` (class in `blatann.nrf.nrf_types.gatt`), 112  
`BLEGattsAttrMetadata` (class in `bla-`

[tann.nrf.nrf\\_types.gatt\), 112](#)  
 BLEGattsAuthorizeParams (class in [blatann.nrf.nrf\\_types.gatt\), 112](#)  
 BLEGattsCharHandles (class in [blatann.nrf.nrf\\_types.gatt\), 112](#)  
 BLEGattsCharMetadata (class in [blatann.nrf.nrf\\_types.gatt\), 112](#)  
 BleGattsEnableParams (class in [blatann.nrf.nrf\\_types.gatt\), 112](#)  
 BLEGattService (class in [blatann.nrf.nrf\\_types.gatt\), 111](#)  
 BLEGattsHvx (class in [blatann.nrf.nrf\\_types.gatt\), 113](#)  
 BLEGattsPresentationFormat (class in [blatann.nrf.nrf\\_types.gatt\), 112](#)  
 BLEGattsRwAuthorizeReplyParams (class in [blatann.nrf.nrf\\_types.gatt\), 113](#)  
 BLEGattStatusCode (class in [blatann.nrf.nrf\\_types.enums\), 106](#)  
 BLEGattValue (class in [blatann.nrf.nrf\\_types.gatt\), 113](#)  
 BLEGattWriteOperation (class in [blatann.nrf.nrf\\_types.enums\), 107](#)  
 BLEGattWriteOperation (class in [blatann.nrf.nrf\\_types.enums\), 105](#)  
 BLEHci (class in [blatann.nrf.nrf\\_types.enums\), 100](#)  
 BleOptConnEventExtention (class in [blatann.nrf.nrf\\_types.config\), 98](#)  
 BleOptGapAuthPayloadTimeout (class in [blatann.nrf.nrf\\_types.config\), 99](#)  
 BleOptGapChannelMap (class in [blatann.nrf.nrf\\_types.config\), 99](#)  
 BleOptGapCompatModel (class in [blatann.nrf.nrf\\_types.config\), 99](#)  
 BleOptGapLocalConnLatency (class in [blatann.nrf.nrf\\_types.config\), 99](#)  
 BleOptGapPasskey (class in [blatann.nrf.nrf\\_types.config\), 99](#)  
 BleOptGapScanRequestReport (class in [blatann.nrf.nrf\\_types.config\), 99](#)  
 BleOptGapSlaveLatencyDisable (class in [blatann.nrf.nrf\\_types.config\), 100](#)  
 BleOption (class in [blatann.nrf.nrf\\_types.config\), 98](#)  
 BleOptionFlag (class in [blatann.nrf.nrf\\_types.config\), 98](#)  
 BleOptPaLna (class in [blatann.nrf.nrf\\_types.config\), 99](#)  
 BlePaLnaConfig (class in [blatann.nrf.nrf\\_types.config\), 98](#)  
 BLEUUID (class in [blatann.nrf.nrf\\_types.generic\), 113](#)  
 BLEUUID.Standard (class in [blatann.nrf.nrf\\_types.generic\), 113](#)  
 BLEUUIDBase (class in [blatann.nrf.nrf\\_types.generic\), 113](#)  
 blood\_pressure (blatann.bt\_sig.uuids.ServiceUuid attribute), 29  
 blood\_pressure (blatann.bt\_sig.uuids.ServiceUuid attribute), 32  
 blood\_pressure\_arm (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 30  
 blood\_pressure\_feature (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35  
 blood\_pressure\_measurement (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35  
 blood\_pressure\_record (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35  
 blood\_pressure\_wrist (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 30  
 bluetooth\_sig (blatann.services.device\_info.data\_types.PnpVendorSource attribute), 128  
 bluetooth\_sig\_data (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35  
 body\_composition (blatann.bt\_sig.uuids.ServiceUuid attribute), 32  
 body\_composition\_feature (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 35  
 body\_composition\_measurement (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
 body\_sensor\_location (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
 bond\_management (blatann.bt\_sig.uuids.ServiceUuid attribute), 32  
 bond\_management\_control\_point (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
 bond\_management\_feature (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
 BondDatabase (class in [blatann.gap.bond\\_db\), 66](#)  
 BondDatabaseLoader (class in [blatann.gap.bond\\_db\), 66](#)  
 BondDbEntry (class in [blatann.gap.bond\\_db\), 66](#)  
 bonded\_device\_repairing (blatann.event\_args.PairingRejectedReason attribute), 156  
 bonded\_peripheral\_request (blatann.event\_args.PairingRejectedReason attribute), 156  
 BONDING (blatann.event\_args.SecurityProcess attribute), 155

BondingData (class in *blatann.gap.bond\_db*), 66  
 boolean (*blatann.bt\_sig.assigned\_numbers.Format* attribute), 23  
 boolean (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 boot\_keyboard\_input\_report (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 boot\_keyboard\_output\_report (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 boot\_mouse\_input\_report (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 bottom (*blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor* attribute), 24  
 BR\_EDR\_CONTROLLER (*blatann.gap.advertise\_data.AdvertisingFlags* attribute), 59  
 br\_edr\_handover\_data (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 BR\_EDR\_HOST (*blatann.gap.advertise\_data.AdvertisingFlags* attribute), 59  
 br\_edr\_in\_prog (*blatann.nrf.nrf\_types.enums.BLEGapSecStatus* attribute), 105  
 BR\_EDR\_NOT\_SUPPORTED (*blatann.gap.advertise\_data.AdvertisingFlags* attribute), 59  
 breakfast (*blatann.services.glucose.data\_types.CarbohydrateType* attribute), 131  
 broadcast\_audio\_announcement (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 32  
 broadcast\_audio\_scan (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 32  
 broadcast\_audio\_scan\_control\_point (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 broadcast\_receive\_state (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 brunch (*blatann.services.glucose.data\_types.CarbohydrateType* attribute), 131  
 bss\_control\_point (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 bss\_response (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 bt\_sig (*blatann.bt\_sig.assigned\_numbers.Namespace* attribute), 24  
 busy (*blatann.nrf.nrf\_types.enums.NrfError* attribute), 102  
 byte\_count (*blatann.services.ble\_data\_types.Int16* attribute), 143  
 byte\_count (*blatann.services.ble\_data\_types.Int32* attribute), 143  
 byte\_count (*blatann.services.ble\_data\_types.Int64* attribute), 143  
 byte\_count (*blatann.services.ble\_data\_types.Int8* attribute), 142  
 byte\_count (*blatann.services.ble\_data\_types.Uint16* attribute), 143  
 byte\_count (*blatann.services.ble\_data\_types.Uint24* attribute), 143  
 byte\_count (*blatann.services.ble\_data\_types.Uint32* attribute), 143  
 byte\_count (*blatann.services.ble\_data\_types.Uint40* attribute), 143  
 byte\_count (*blatann.services.ble\_data\_types.Uint48* attribute), 143  
 byte\_count (*blatann.services.ble\_data\_types.Uint56* attribute), 143  
 byte\_count (*blatann.services.ble\_data\_types.Uint64* attribute), 143  
 byte\_count (*blatann.services.ble\_data\_types.Uint8* attribute), 142  
 byte\_count (*blatann.services.ble\_data\_types.UnsignedIntegerBase* attribute), 142  
 byte\_count() (*blatann.services.ble\_data\_types.Bitfield* class method), 145  
 bytes\_per\_notification (*blatann.peer.Peer* property), 160  
**C**  
 call\_control\_point (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 call\_control\_point\_optional\_opcodes (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 call\_friendly\_name (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 call\_state (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 caloric\_intake (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 36  
 can\_enable\_notifications (*blatann.services.battery.service.BatteryClient* property), 122  
 can\_enable\_notifications (*blatann.services.current\_time.service.CurrentTimeClient* property), 127



`can_set_current_time` (blatann.services.current\_time.service.CurrentTimeCharacteristic property), 127  
`can_set_local_time_info` (blatann.services.current\_time.service.CurrentTimeCharacteristic property), 127  
`capacitance_farad` (blatann.bt\_sig.assigned\_numbers.Units attribute), 25  
`capillary_plasma` (blatann.services.glucose.data\_types.GlucoseType attribute), 130  
`capillary_whole_blood` (blatann.services.glucose.data\_types.GlucoseType attribute), 130  
`CarbohydrateType` (class in blatann.services.glucose.data\_types), 131  
`carbon_monoxide_concentration` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`CarbsInfo` (class in blatann.services.glucose.data\_types), 134  
`cardiorespiratory_activity_instantaneous_data` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`cardiorespiratory_activity_summary_data` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`casual` (blatann.services.glucose.data\_types.MealType attribute), 131  
`catalytic_activity_concentration_katal_per_cubic_metre` (blatann.bt\_sig.assigned\_numbers.Units attribute), 25  
`catalytic_activity_katal` (blatann.bt\_sig.assigned\_numbers.Units attribute), 25  
`cccd` (blatann.bt\_sig.uuids.DescriptorUuid attribute), 31  
`cccd` (blatann.nrf.nrf\_types.generic.BLEUUID.Standard attribute), 113  
`cellular_network` (blatann.services.current\_time.data\_types.TimeSource attribute), 124  
`central` (blatann.nrf.nrf\_types.enums.BLEGapRoles attribute), 104  
`central_address_resolution` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`cgm_feature` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`cgm_measurement` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`cgm_session_run_time` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`cgm_session_start_time` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`cgm_specific_ops_control_point` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`cgm_status` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`char_add()` (blatann.nrf.nrf\_types.gatt.BLEGattService method), 111  
`char_array_to_list()` (in module blatann.nrf.nrf\_driver\_types), 119  
`char_uuid` (blatann.nrf.nrf\_types.gatt.BLEGattCharacteristic attribute), 111  
`characteristic` (blatann.bt\_sig.uuids.DeclarationUuid attribute), 31  
`characteristic` (blatann.nrf.nrf\_types.generic.BLEUUID.Standard attribute), 113  
`Characteristic` (class in blatann.gatt), 77  
`CharacteristicProperties` (class in blatann.gatt), 77  
`characteristics` (blatann.gatt.gattc.GattcService property), 81  
`characteristics` (blatann.gatt.gatts.GattsService property), 87  
`CharacteristicUuid` (class in blatann.bt\_sig.uuids), 34  
`check_encoded_length()` (blatann.gap.advertise\_data.AdvertisingData method), 61  
`chromatic_distance_from_planckian` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`chromaticity_coordinate` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`chromaticity_coordinates` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 36  
`chromaticity_in_cct_and_duv_values` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 37  
`chromaticity_tolerance` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 37  
`cie_color_rendering_index` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 37  
`class_of_device` (blatann.gap.advertise\_data.AdvertisingData.Types attribute), 60  
`class_of_device` (blatann.gap.advertise\_data.AdvertisingData attribute), 60

`tann.nrf.nrf_types.gap.BLEAdvData.Types`  
`attribute`), 109  
`clear()` (`blatann.gap.advertise_data.ScanReportCollection` `complete_local_name` `method`), 63  
`clear_all()` (`blatann.gatt.managers.GattcOperationManager` `method`), 90  
`clear_all()` (`blatann.gatt.managers.GattsOperationManager` `method`), 90  
`clear_all()` (`blatann.utils.queued_tasks_manager.QueuedTasksManager` `method`), 147  
`clear_bonding_data()` (`blatann.device.BleDevice` `method`), 152  
`clear_handlers()` (`blatann.event_type.EventSource` `method`), 158  
`clear_pending_notifications()` (`blatann.gatt.gatts.GattsDatabase` `method`), 88  
`Client` (*class in* `blatann.peer`), 164  
`CLIENT_DISCONNECTED` (`blatann.event_args.GattOperationCompleteReason` `attribute`), 154  
`client_subscribed` (`blatann.gatt.gatts.GattsCharacteristic` `property`), 86  
`client_supported_features` (`blatann.bt_sig.uuids.CharacteristicUuid` `attribute`), 37  
`CLIENT_UNSUBSCRIBED` (`blatann.event_args.GattOperationCompleteReason` `attribute`), 154  
`ClientConnectionWaitable` (*class in* `blatann.waitables.connection_waitable`), 147  
`clock` (`blatann.bt_sig.assigned_numbers.Appearance` `attribute`), 29  
`close()` (`blatann.device.BleDevice` `method`), 152  
`close()` (`blatann.nrf.nrf_driver.NrfDriver` `method`), 117  
`coded` (`blatann.nrf.nrf_types.enums.BLEGapPhy` `attribute`), 104  
`coefficient` (`blatann.bt_sig.uuids.CharacteristicUuid` `attribute`), 37  
`combine()` (`blatann.gap.smp.PairingPolicy` `static method`), 72  
`combine_with_base()` (`blatann.uuid.Uuid128` `class method`), 165  
`command_disallowed` (`blatann.nrf.nrf_types.enums.BLEHci` `attribute`), 101  
`common_audio` (`blatann.bt_sig.uuids.ServiceUuid` `attribute`), 32  
`company_assigned_uuid16s` (*in module* `blatann.bt_sig.uuids`), 47  
`complete_br_edr_transport_block_data` (`blatann.bt_sig.uuids.DescriptorUuid` `attribute`), 32  
`complete_local_name` (`blatann.gap.advertise_data.AdvertisingData.Types` `attribute`), 60  
`computer` (`blatann.bt_sig.assigned_numbers.Appearance` `attribute`), 29  
`concentration_count_per_cubic_metre` (`blatann.bt_sig.assigned_numbers.Units` `attribute`), 25  
`concentration_parts_per_billion` (`blatann.bt_sig.assigned_numbers.Units` `attribute`), 25  
`concentration_parts_per_million` (`blatann.bt_sig.assigned_numbers.Units` `attribute`), 25  
`configure()` (`blatann.device.BleDevice` `method`), 151  
`configure_automatic()` (`blatann.services.current_time.service.CurrentTimeServer` `method`), 126  
`confirm_value` (`blatann.nrf.nrf_types.enums.BLEGapSecStatus` `attribute`), 105  
`conn` (`blatann.nrf.nrf_types.enums.BLEGapTimeoutSrc` `attribute`), 104  
`conn_count` (`blatann.nrf.nrf_types.enums.NrfError` `attribute`), 102  
`conn_event_extension` (`blatann.nrf.nrf_types.config.BleOptionFlag` `attribute`), 98  
`conn_failed_to_be_established` (`blatann.nrf.nrf_types.enums.BLEHci` `attribute`), 101  
`conn_interval_unacceptable` (`blatann.nrf.nrf_types.enums.BLEHci` `attribute`), 101  
`conn_params_setup()` (`blatann.nrf.nrf_driver.NrfDriver` `method`), 117  
`conn_terminated_due_to_mic_failure` (`blatann.nrf.nrf_types.enums.BLEHci` `attribute`), 101  
`connect()` (`blatann.device.BleDevice` `method`), 153  
`connectable_directed` (`blatann.nrf.nrf_types.enums.BLEGapAdvType` `attribute`), 104  
`connectable_undirected` (`blatann.nrf.nrf_types.enums.BLEGapAdvType` `attribute`), 104  
`connected` (`blatann.peer.Peer` `property`), 160  
`CONNECTED` (`blatann.peer.PeerState` `attribute`), 159  
`CONNECTING` (`blatann.peer.PeerState` `attribute`), 159  
`connection_timeout` (`blatann.nrf.nrf_types.enums.BLEHci` `attribute`), 101  
`ConnectionManager` (*class in* `bla-`

- tann.examples.central\_event\_driven*), 52
- `ConnectionParameters` (class in *blatann.gap.gap\_types*), 70
- `ConnectionParametersUpdatedEventArgs` (class in *blatann.event\_args*), 155
- `ConnectionWaitable` (class in *blatann.waitables.connection\_waitable*), 147
- `constant_tone_extension` (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 32
- `constant_tone_extension_enable` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `content_control_id` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `continuous_glucose_monitoring` (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 32
- `control_solution` (*blatann.services.glucose.data\_types.GlucoseType* attribute), 131
- `control_solution` (*blatann.services.glucose.data\_types.SampleLocation* attribute), 131
- `controller_busy` (*blatann.nrf.nrf\_types.enums.BLEHci* attribute), 101
- `coordinated_set_identification` (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 32
- `coordinated_set_size` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `correlated_color_temperature` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `count_16` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `count_24` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `CountingCharacteristicThread` (class in *blatann.examples.peripheral*), 54
- `country_code` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `cps_cccd_config_error` (*blatann.nrf.nrf\_types.enums.BLEGattStatusCode* attribute), 107
- `cps_out_of_range` (*blatann.nrf.nrf\_types.enums.BLEGattStatusCode* attribute), 107
- `cps_proc_alr_in_prog` (*blatann.nrf.nrf\_types.enums.BLEGattStatusCode* attribute), 107
- `create()` (*blatann.gap.bond\_db.BondDatabase* method), 66
- `create()` (*blatann.gap.default\_bond\_db.DefaultBondDatabase* method), 68
- `cross_trainer_data` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `csc_feature` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `csc_measurement` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `current_density_ampere_per_square_metre` (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 25
- `current_group_object_id` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `current_time` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `current_time` (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 32
- `current_track_object_id` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `current_track_segments_object_id` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `CurrentTime` (class in *blatann.services.current\_time.data\_types*), 124
- `CurrentTimeClient` (class in *blatann.services.current\_time.service*), 127
- `CurrentTimeServer` (class in *blatann.services.current\_time.service*), 125
- `cycling` (*blatann.bt\_sig.assigned\_numbers.Appearance* attribute), 30
- `cycling_cadence_sensor` (*blatann.bt\_sig.assigned\_numbers.Appearance* attribute), 30
- `cycling_cycling_computer` (*blatann.bt\_sig.assigned\_numbers.Appearance* attribute), 30
- `cycling_power` (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 32
- `cycling_power_control_point` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `cycling_power_feature` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `cycling_power_measurement` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- `cycling_power_sensor` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37

- tann.bt\_sig.assigned\_numbers.Appearance* attribute), 30
- cycling\_power\_vector (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- cycling\_speed\_and\_cadence (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 32
- cycling\_speed\_cadence\_sensor (*blatann.bt\_sig.assigned\_numbers.Appearance* attribute), 30
- cycling\_speed\_sensor (*blatann.bt\_sig.assigned\_numbers.Appearance* attribute), 30
- ## D
- data\_size (*blatann.nrf.nrf\_types.enums.NrfError* attribute), 102
- data\_stream\_types (*blatann.gatt.PresentationFormat* attribute), 78
- data\_stream\_types (*blatann.services.ble\_data\_types.BleCompoundDataType* attribute), 141
- data\_stream\_types (*blatann.services.ble\_data\_types.DateTime* attribute), 144
- data\_stream\_types (*blatann.services.ble\_data\_types.DayDateTime* attribute), 145
- data\_stream\_types (*blatann.services.current\_time.data\_types.CurrentTime* attribute), 124
- data\_stream\_types (*blatann.services.current\_time.data\_types.ExactTime256* attribute), 124
- data\_stream\_types (*blatann.services.current\_time.data\_types.LocalTimeInfo* attribute), 125
- data\_stream\_types (*blatann.services.current\_time.data\_types.ReferenceTimeInfo* attribute), 125
- data\_stream\_types (*blatann.services.device\_info.data\_types.PnpId* attribute), 128
- data\_stream\_types (*blatann.services.device\_info.data\_types.SystemId* attribute), 128
- database (*blatann.device.BleDevice* property), 152
- database (*blatann.peer.Peer* property), 161
- database\_change\_increment (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- database\_hash (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- database\_strategies (in module *blatann.gap.default\_bond\_db*), 68
- database\_strategies\_by\_extension (in module *blatann.gap.default\_bond\_db*), 68
- DatabaseDiscoverer (class in *blatann.gatt.service\_discovery*), 91
- DatabaseDiscoveryCompleteEventArgs (class in *blatann.event\_args*), 157
- DatabaseStrategy (class in *blatann.gap.default\_bond\_db*), 67
- DataLengthUpdatedEventArgs (class in *blatann.event\_args*), 155
- date\_of\_birth (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- date\_of\_threshold\_assessment (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- date\_time (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- date\_utc (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- DateTime (class in *blatann.services.ble\_data\_types*), 144
- day\_date\_time (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- day\_of\_week (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 37
- DayDateTime (class in *blatann.services.ble\_data\_types*), 145
- DaylightSavingsTimeOffset (class in *blatann.services.current\_time.data\_types*), 123
- DayOfWeek (class in *blatann.services.ble\_data\_types*), 144
- declaration\_attribute (*blatann.gatt.gattc.GattcCharacteristic* property), 78
- DeclarationUuid (class in *blatann.bt\_sig.uuids*), 31
- decode() (*blatann.gatt.PresentationFormat* class method), 78
- decode() (*blatann.services.ble\_data\_types.Bitfield* class method), 145
- decode() (*blatann.services.ble\_data\_types.BleCompoundDataType* class method), 141
- decode() (*blatann.services.ble\_data\_types.BleDataStream* method), 141
- decode() (*blatann.services.ble\_data\_types.BleDataType* class method), 142
- decode() (*blatann.services.ble\_data\_types.DateTime* class method), 144
- decode() (*blatann.services.ble\_data\_types.DayDateTime* class method), 145
- decode() (*blatann.services.ble\_data\_types.DoubleNibble* class method), 142
- decode() (*blatann.services.ble\_data\_types.SFloat* class method), 144

`decode()` (*blatann.services.ble\_data\_types.String* class method), 143  
`decode()` (*blatann.services.ble\_data\_types.UnsignedIntegerBase* class method), 142  
`decode()` (*blatann.services.current\_time.data\_types.CurrentTime* class method), 124  
`decode()` (*blatann.services.current\_time.data\_types.ExactTime256* class method), 124  
`decode()` (*blatann.services.current\_time.data\_types.LocalTimeInfo* class method), 125  
`decode()` (*blatann.services.current\_time.data\_types.ReferenceTimeInfo* class method), 125  
`decode()` (*blatann.services.decoded\_event\_dispatcher.DecodedReadWriteEventDispatcher* class method), 146  
`decode()` (*blatann.services.device\_info.data\_types.PnpId* class method), 128  
`decode()` (*blatann.services.device\_info.data\_types.SystemId* class method), 128  
`decode()` (*blatann.services.glucose.data\_types.CarbsInfo* class method), 134  
`decode()` (*blatann.services.glucose.data\_types.ExerciseInfo* class method), 135  
`decode()` (*blatann.services.glucose.data\_types.GlucoseContext* class method), 135  
`decode()` (*blatann.services.glucose.data\_types.GlucoseMeasurement* class method), 134  
`decode()` (*blatann.services.glucose.data\_types.GlucoseSample* class method), 134  
`decode()` (*blatann.services.glucose.data\_types.MedicationInfo* class method), 135  
`decode()` (*blatann.services.glucose.racp.RacpCommand* class method), 138  
`decode()` (*blatann.services.glucose.racp.RacpResponse* class method), 139  
`decode_if()` (*blatann.services.ble\_data\_types.BleDataStream* method), 141  
`decode_if_multiple()` (*blatann.services.ble\_data\_types.BleDataStream* method), 141  
`decode_multiple()` (*blatann.services.ble\_data\_types.BleDataStream* method), 141  
`DecodedReadCompleteEventArgs` (class in *blatann.event\_args*), 157  
`DecodedReadWriteEventDispatcher` (class in *blatann.services.decoded\_event\_dispatcher*), 146  
`DecodedWriteEventArgs` (class in *blatann.event\_args*), 156  
`DecodeError`, 159  
`default_baud_rate` (*blatann.nrf.nrf\_driver.NrfDriver* attribute), 117  
`DEFAULT_CONN_TAG` (*blatann.nrf.nrf\_types.config.BleConnConfig* attribute), 100  
`DEFAULT_PRIVATE_ADDR_CYCLE_INTERVAL_S` (*blatann.nrf.nrf\_types.gap.BLEGapPrivacyParams* attribute), 110  
`DefaultBondDatabase` (class in *blatann.gap.default\_bond\_db*), 68  
`DefaultBondDatabaseLoader` (class in *blatann.gap.default\_bond\_db*), 68  
`delete()` (*blatann.gap.bond\_db.BondDatabase* method), 66  
`delete()` (*blatann.gap.default\_bond\_db.DefaultBondDatabase* method), 69  
`delete_all()` (*blatann.gap.bond\_db.BondDatabase* method), 69  
`delete_all()` (*blatann.gap.default\_bond\_db.DefaultBondDatabase* method), 69  
`delete_bonding_data()` (*blatann.gap.smp.SecurityManager* method), 75  
`delete_records()` (*blatann.services.glucose.database.BasicGlucoseDatabase* method), 137  
`delete_records()` (*blatann.services.glucose.database.IGlucoseDatabase* method), 136  
`delete_stored_records()` (*blatann.services.glucose.racp.RacpOpcode* attribute), 137  
`density_kilogram_per_cubic_metre` (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 26  
`deregister()` (*blatann.event\_type.Event* method), 158  
`desc_array_to_list()` (in module *blatann.nrf.nrf\_driver\_types*), 119  
`description` (*blatann.utils.IntEnumWithDescription* property), 146  
`descriptive_string` (*blatann.uuid.Uuid* property), 165  
`descriptor_value_changed` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 38  
`DescriptorUuid` (class in *blatann.bt\_sig.uuids*), 31  
`device_information` (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 32  
`device_name` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 38  
`device_name` (*blatann.gap.advertise\_data.ScanReport* property), 62  
`device_name` (*blatann.gap.generic\_access\_service.GenericAccessService* property), 70  
`DEVICE_NAME_MAX_LENGTH` (*blatann.gap.generic\_access\_service.GenericAccessService* attribute), 70  
`device_time` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 38



- attribute), 38
- device\_time (blatann.bt\_sig.uuids.ServiceUuid attribute), 32
- device\_time\_control\_point (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38
- device\_time\_feature (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38
- device\_time\_parameters (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38
- device\_wearing\_position (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38
- dew\_point (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38
- dhkey\_failure (blatann.nrf.nrf\_types.enums.BLEGapSecurityStatus attribute), 105
- differen\_transaction\_collision (blatann.nrf.nrf\_types.enums.BLEHci attribute), 101
- digital (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38
- digital\_output (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38
- dinner (blatann.services.glucose.data\_types.CarbohydrateType attribute), 131
- directed\_advertiser\_timeout (blatann.nrf.nrf\_types.enums.BLEHci attribute), 101
- directory\_listing (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38
- disable\_notifications() (blatann.services.battery.service.BatteryClient method), 122
- DisClient (class in blatann.services.device\_info.service), 129
- disconnect() (blatann.peer.Peer method), 162
- DISCONNECTED (blatann.peer.PeerState attribute), 159
- DisconnectionEventArgs (class in blatann.event\_args), 154
- DisconnectionWaitable (class in blatann.waitables.connection\_waitable), 147
- discover\_services() (blatann.peer.Peer method), 163
- discovered\_handles() (blatann.nrf.nrf\_types.gatt.BLEGattCharacteristic method), 111
- display (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 29
- DISPLAY\_ONLY (blatann.nrf.nrf\_types.enums.BLEGapIoCaps attribute), 104
- display\_passkey() (in module blatann.examples.peripheral\_glucose\_service), 57
- DISPLAY\_YESNO (blatann.nrf.nrf\_types.enums.BLEGapIoCaps attribute), 104
- DisServer (class in blatann.services.device\_info.service), 129
- DLE\_OVERHEAD (in module blatann.gatt), 76
- DLE\_SIZE\_DEFAULT (in module blatann.gap), 59
- DLE\_SIZE\_MINIMUM (in module blatann.gap), 59
- dose\_equivalent\_sievert (blatann.bt\_sig.assigned\_numbers.Units attribute), 26
- DoubleNibble (class in blatann.services.ble\_data\_types), 142
- drink (blatann.services.glucose.data\_types.CarbohydrateType attribute), 131
- status\_change (blatann.services.current\_time.data\_types.AdjustmentReasonType attribute), 123
- dst\_offset (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38
- duint16 (blatann.bt\_sig.assigned\_numbers.Format attribute), 24
- during\_menses (blatann.services.glucose.data\_types.HealthStatus attribute), 132
- dynamic\_viscosity\_pascal\_second (blatann.bt\_sig.assigned\_numbers.Units attribute), 26
- ## E
- earlobe (blatann.services.glucose.data\_types.SampleLocation attribute), 131
- elapsed (blatann.utils.Stopwatch property), 146
- electric\_charge\_ampere\_hours (blatann.bt\_sig.assigned\_numbers.Units attribute), 26
- electric\_charge\_coulomb (blatann.bt\_sig.assigned\_numbers.Units attribute), 26
- electric\_charge\_density\_coulomb\_per\_cubic\_metre (blatann.bt\_sig.assigned\_numbers.Units attribute), 26
- electric\_conductance\_siemens (blatann.bt\_sig.assigned\_numbers.Units attribute), 26
- electric\_current (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38
- electric\_current\_ampere (blatann.bt\_sig.assigned\_numbers.Units attribute), 26
- electric\_current\_range (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38

<code>electric_current_specification</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 38	<code>encode()</code> ( <code>blatann.services.ble_data_types.String</code> class method), 143
<code>electric_current_statistics</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 38	<code>encode()</code> ( <code>blatann.services.ble_data_types.UnsignedIntegerBase</code> class method), 142
<code>electric_field_strength_volt_per_metre</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 26	<code>encode()</code> ( <code>blatann.services.current_time.data_types.CurrentTime</code> method), 124
<code>electric_flux_density_coulomb_per_square_metre</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 26	<code>encode()</code> ( <code>blatann.services.current_time.data_types.ExactTime256</code> method), 124
<code>electric_potential_difference_volt</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 26	<code>encode()</code> ( <code>blatann.services.current_time.data_types.LocalTimeInfo</code> method), 125
<code>electric_resistance_ohm</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 26	<code>encode()</code> ( <code>blatann.services.current_time.data_types.ReferenceTimeInfo</code> method), 125
<code>elevation</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 38	<code>encode()</code> ( <code>blatann.services.device_info.data_types.PnpId</code> method), 128
<code>email_address</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 38	<code>encode()</code> ( <code>blatann.services.device_info.data_types.SystemId</code> method), 128
<code>emergency_configuration</code> ( <code>blatann.bt_sig.uuids.ServiceUuid</code> attribute), 32	<code>encode()</code> ( <code>blatann.services.glucose.data_types.CarbsInfo</code> method), 134
<code>emergency_id</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 38	<code>encode()</code> ( <code>blatann.services.glucose.data_types.ExerciseInfo</code> method), 135
<code>emergency_text</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 38	<code>encode()</code> ( <code>blatann.services.glucose.data_types.GlucoseContext</code> method), 135
<code>EmptyWaitable</code> (class in <code>blatann.waitables.waitable</code> ), 150	<code>encode()</code> ( <code>blatann.services.glucose.data_types.GlucoseMeasurement</code> method), 134
<code>enable_notifications()</code> ( <code>blatann.services.battery.service.BatteryClient</code> method), 122	<code>encode()</code> ( <code>blatann.services.glucose.data_types.GlucoseSample</code> method), 134
<code>enc_key_size</code> ( <code>blatann.nrf.nrf_types.enums.BLEGapSecStatus</code> attribute), 105	<code>encode()</code> ( <code>blatann.services.glucose.data_types.MedicationInfo</code> method), 135
<code>encode()</code> ( <code>blatann.gatt.PresentationFormat</code> method), 78	<code>encode()</code> ( <code>blatann.services.glucose.racp.RacpCommand</code> method), 138
<code>encode()</code> ( <code>blatann.services.ble_data_types.Bitfield</code> method), 145	<code>encode()</code> ( <code>blatann.services.glucose.racp.RacpResponse</code> method), 139
<code>encode()</code> ( <code>blatann.services.ble_data_types.BleCompoundDataType</code> method), 141	<code>encode_if()</code> ( <code>blatann.services.ble_data_types.BleDataStream</code> method), 141
<code>encode()</code> ( <code>blatann.services.ble_data_types.BleDataStream</code> method), 141	<code>encode_if_multiple()</code> ( <code>blatann.services.ble_data_types.BleDataStream</code> method), 141
<code>encode()</code> ( <code>blatann.services.ble_data_types.BleDataType</code> class method), 142	<code>encode_multiple()</code> ( <code>blatann.services.ble_data_types.BleDataStream</code> method), 141
<code>encode()</code> ( <code>blatann.services.ble_data_types.DateTime</code> method), 144	<code>encode_values()</code> ( <code>blatann.services.ble_data_types.BleCompoundDataType</code> method), 141
<code>encode()</code> ( <code>blatann.services.ble_data_types.DayDateTime</code> method), 145	<code>encoded_size()</code> ( <code>blatann.services.ble_data_types.Bitfield</code> class method), 145
<code>encode()</code> ( <code>blatann.services.ble_data_types.DoubleNibble</code> class method), 142	<code>encoded_size()</code> ( <code>blatann.services.ble_data_types.BleCompoundDataType</code> class method), 142
<code>encode()</code> ( <code>blatann.services.ble_data_types.SFloat</code> class method), 144	<code>encoded_size()</code> ( <code>blatann.services.ble_data_types.BleDataType</code> class method), 142
	<code>encoded_size()</code> ( <code>blatann.services.ble_data_types.DoubleNibble</code> class method), 142

class method), 142  
 encoded\_size() (blatann.services.ble\_data\_types.SFloat class method), 144  
 encoded\_size() (blatann.services.ble\_data\_types.UnsignedIntegerBase class method), 142  
 ENCRYPTION (blatann.event\_args.SecurityProcess attribute), 155  
 ENCRYPTION (blatann.nrf.nrf\_types.smp.BLEGapSecModeType attribute), 114  
 energy (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38  
 energy\_density\_joule\_per\_cubic\_metre (blatann.bt\_sig.assigned\_numbers.Units attribute), 26  
 energy\_gram\_calorie (blatann.bt\_sig.assigned\_numbers.Units attribute), 26  
 energy\_in\_a\_period\_of\_day (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38  
 energy\_joule (blatann.bt\_sig.assigned\_numbers.Units attribute), 26  
 energy\_kilogram\_calorie (blatann.bt\_sig.assigned\_numbers.Units attribute), 26  
 energy\_kilowatt\_hour (blatann.bt\_sig.assigned\_numbers.Units attribute), 26  
 enhanced\_blood\_pressure\_measurement (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38  
 enhanced\_intermediate\_cuff\_pressure (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38  
 environmental\_sensing (blatann.bt\_sig.uuids.ServiceUuid attribute), 32  
 es\_configuration (blatann.bt\_sig.uuids.DescriptorUuid attribute), 31  
 es\_measurement (blatann.bt\_sig.uuids.DescriptorUuid attribute), 31  
 es\_trigger\_setting (blatann.bt\_sig.uuids.DescriptorUuid attribute), 31  
 Event (class in blatann.event\_type), 158  
 event\_decode() (in module blatann.nrf.nrf\_events), 92  
 event\_statistics (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 38  
 event\_subscribe() (blatann.nrf.nrf\_driver.NrfDriver method), 117  
 event\_unsubscribe() (blatann.nrf.nrf\_driver.NrfDriver method), 117  
 event\_unsubscribe\_all() (blatann.nrf.nrf\_driver.NrfDriver method), 117  
 EventArgs (class in blatann.event\_args), 154  
 EventSource (class in blatann.event\_type), 158  
 EventSubscriptionContext (class in blatann.event\_type), 158  
 EventWaitable (class in blatann.waitables.event\_waitable), 148  
 evt\_id (blatann.nrf.nrf\_events.gap\_events.GapEvtAdvReport attribute), 92  
 evt\_id (blatann.nrf.nrf\_events.gap\_events.GapEvtConnected attribute), 93  
 evt\_id (blatann.nrf.nrf\_events.gap\_events.GapEvtConnParamUpdate attribute), 92  
 evt\_id (blatann.nrf.nrf\_events.gap\_events.GapEvtConnParamUpdateRequest attribute), 92  
 evt\_id (blatann.nrf.nrf\_events.gap\_events.GapEvtDataLengthUpdate attribute), 93  
 evt\_id (blatann.nrf.nrf\_events.gap\_events.GapEvtDataLengthUpdateRequest attribute), 93  
 evt\_id (blatann.nrf.nrf\_events.gap\_events.GapEvtDisconnected attribute), 93  
 evt\_id (blatann.nrf.nrf\_events.gap\_events.GapEvtPhyUpdate attribute), 93  
 evt\_id (blatann.nrf.nrf\_events.gap\_events.GapEvtPhyUpdateRequest attribute), 93  
 evt\_id (blatann.nrf.nrf\_events.gap\_events.GapEvtRssiChanged attribute), 92  
 evt\_id (blatann.nrf.nrf\_events.gap\_events.GapEvtTimeout attribute), 92  
 evt\_id (blatann.nrf.nrf\_events.gatt\_events.GattcEvtAttrInfoDiscoveryResponse attribute), 95  
 evt\_id (blatann.nrf.nrf\_events.gatt\_events.GattcEvtCharacteristicDiscoveryRequest attribute), 94  
 evt\_id (blatann.nrf.nrf\_events.gatt\_events.GattcEvtDescriptorDiscoveryRequest attribute), 95  
 evt\_id (blatann.nrf.nrf\_events.gatt\_events.GattcEvtHvx attribute), 94  
 evt\_id (blatann.nrf.nrf\_events.gatt\_events.GattcEvtMtuExchangeResponse attribute), 95  
 evt\_id (blatann.nrf.nrf\_events.gatt\_events.GattcEvtPrimaryServiceDiscoveryRequest attribute), 94  
 evt\_id (blatann.nrf.nrf\_events.gatt\_events.GattcEvtReadResponse attribute), 94  
 evt\_id (blatann.nrf.nrf\_events.gatt\_events.GattcEvtTimeout attribute), 95  
 evt\_id (blatann.nrf.nrf\_events.gatt\_events.GattcEvtWriteCmdTxComplete attribute), 94  
 evt\_id (blatann.nrf.nrf\_events.gatt\_events.GattcEvtWriteResponse attribute), 94  
 evt\_id (blatann.nrf.nrf\_events.gatt\_events.GattsEvtExchangeMtuRequest attribute), 96



evt\_id(*blatann.nrf.nrf\_events.gatt\_events.GattsEvtHandleValueConf* attribute), 135  
 attribute), 96 ExerciseInfo (class in bla-  
 evt\_id(*blatann.nrf.nrf\_events.gatt\_events.GattsEvtNotificationTxComplete* services.glucose.data\_types), 134  
 attribute), 96 exposure\_coulomb\_per\_kilogram (bla-  
 evt\_id(*blatann.nrf.nrf\_events.gatt\_events.GattsEvtReadWriteAuthorizationRequest* assigned\_numbers.Units attribute),  
 attribute), 96 26  
 evt\_id(*blatann.nrf.nrf\_events.gatt\_events.GattsEvtSysAttr* Extended\_properties (bla-  
 attribute), 95 tann.bt\_sig.uuids.DescriptorUuid attribute),  
 evt\_id(*blatann.nrf.nrf\_events.gatt\_events.GattsEvtTimeout* 31  
 attribute), 96 external(*blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor*  
 attribute), 24  
 evt\_id(*blatann.nrf.nrf\_events.gatt\_events.GattsEvtWrite* external\_report\_reference (bla-  
 attribute), 95  
 evt\_id(*blatann.nrf.nrf\_events.generic\_events.BLEEvent* tann.bt\_sig.uuids.DescriptorUuid attribute),  
 attribute), 96 31  
 evt\_id(*blatann.nrf.nrf\_events.generic\_events.EvtUserMemoryRequest* external\_time\_reference\_update (bla-  
 attribute), 96 tann.services.current\_time.data\_types.AdjustmentReasonType  
 evt\_id(*blatann.nrf.nrf\_events.smp\_events.GapEvtAuthKeyRequest* attribute), 123  
 attribute), 97 eye\_glasses(*blatann.bt\_sig.assigned\_numbers.Appearance*  
 evt\_id(*blatann.nrf.nrf\_events.smp\_events.GapEvtAuthStatus* attribute), 29  
 attribute), 97  
 evt\_id(*blatann.nrf.nrf\_events.smp\_events.GapEvtConnSecUpdate* FAILED(*blatann.event\_args.GattOperationCompleteReason*  
 attribute), 97 attribute), 154  
 evt\_id(*blatann.nrf.nrf\_events.smp\_events.GapEvtLescDhKeyRequest* attribute), 154  
 attribute), 98 fasting(*blatann.services.glucose.data\_types.MealType*  
 evt\_id(*blatann.nrf.nrf\_events.smp\_events.GapEvtPasskeyDisplay* attribute), 131  
 attribute), 97 fat\_burn\_heart\_rate\_lower\_limit (bla-  
 evt\_id(*blatann.nrf.nrf\_events.smp\_events.GapEvtSecInfoRequest* tann.bt\_sig.uuids.CharacteristicUuid at-  
 attribute), 97 tribute), 38  
 evt\_id(*blatann.nrf.nrf\_events.smp\_events.GapEvtSecParamsRequest* fat\_burn\_heart\_rate\_upper\_limit (bla-  
 attribute), 97 tann.bt\_sig.uuids.CharacteristicUuid at-  
 evt\_id(*blatann.nrf.nrf\_events.smp\_events.GapEvtSecRequest* tribute), 38  
 attribute), 97 file\_extension (bla-  
 EvtUserMemoryRequest (class in bla- tann.gap.default\_bond\_db.DatabaseStrategy  
 tann.nrf.nrf\_events.generic\_events), 96 property), 67  
 exact\_time\_100 (bla- file\_extension (bla-  
 tann.bt\_sig.uuids.CharacteristicUuid at- tann.gap.default\_bond\_db.JsonDatabaseStrategy  
 tribute), 38 property), 67  
 exact\_time\_256 (bla- file\_extension (bla-  
 tann.bt\_sig.uuids.CharacteristicUuid at- tann.gap.default\_bond\_db.PickleDatabaseStrategy  
 tribute), 38 property), 68  
 ExactTime256 (class in bla- FilterType (class in *blatann.services.glucose.racp*),  
 tann.services.current\_time.data\_types), 124 138  
 exchange\_mtu() (*blatann.peer.Peer* method), 162 find\_battery\_service() (in module bla-  
 exec\_write\_req\_cancel (bla- tann.services.battery), 121  
 tann.nrf.nrf\_types.enums.BLEGattsWriteOperation find\_characteristic() (bla-  
 attribute), 107 tann.gatt.gattc.GattcDatabase method),  
 exec\_write\_req\_now (bla- 82  
 tann.nrf.nrf\_types.enums.BLEGattsWriteOperation find\_characteristic() (bla-  
 attribute), 107 tann.gatt.gattc.GattcService method), 81  
 execute\_write\_req (bla- find\_descriptor() (bla-  
 tann.nrf.nrf\_types.enums.BLEGattWriteOperation tann.gatt.gattc.GattcCharacteristic method),  
 attribute), 106 81  
 EXERCISE\_DURATION\_OVERRUN (bla- find\_device\_info\_service() (in module bla-  
 tann.services.glucose.data\_types.ExerciseInfo tann.services.device\_info), 127

<code>find_entry()</code> ( <code>blatann.gap.bond_db.BondDatabase</code> method), 66	<code>fixed_string_36</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39
<code>find_entry()</code> ( <code>blatann.gap.default_bond_db.DefaultBondDatabase</code> method), 69	<code>fixed_string_8</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39
<code>find_in_database()</code> ( <code>blatann.services.battery.service.BatteryClient</code> class method), 122	<code>flags</code> ( <code>blatann.gap.advertise_data.AdvertisingData</code> property), 61
<code>find_in_database()</code> ( <code>blatann.services.device_info.service.DisClient</code> class method), 129	<code>flags</code> ( <code>blatann.gap.advertise_data.AdvertisingData.Types</code> attribute), 60
<code>find_in_database()</code> ( <code>blatann.services.nordic_uart.service.NordicUartClient</code> class method), 141	<code>flags</code> ( <code>blatann.nrf.nrf_types.gap.BLEAdvData.Types</code> attribute), 109
<code>find_nordic_uart_service()</code> (in module <code>blatann.services.nordic_uart</code> ), 140	<code>flash</code> ( <code>blatann.bt_sig.assigned_numbers.NamespaceDescriptor</code> attribute), 24
<code>find_service()</code> ( <code>blatann.gatt.gattc.GattcDatabase</code> method), 82	<code>float</code> ( <code>blatann.bt_sig.assigned_numbers.Format</code> attribute), 24
<code>find_target_device()</code> (in module <code>blatann.examples.example_utils</code> ), 52	<code>float32</code> ( <code>blatann.bt_sig.assigned_numbers.Format</code> attribute), 24
<code>finger</code> ( <code>blatann.services.glucose.data_types.SampleLocation</code> attribute), 131	<code>float64</code> ( <code>blatann.bt_sig.assigned_numbers.Format</code> attribute), 24
<code>firmware_revision_string</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 38	<code>floor_number</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39
<code>first_name</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 38	<code>forbidden</code> ( <code>blatann.nrf.nrf_types.enums.NrfError</code> attribute), 102
<code>first_record</code> ( <code>blatann.services.glucose.racp.RacpOperat</code> attribute), 138	<code>force_newton</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 26
<code>first_record()</code> ( <code>blatann.services.glucose.database.BasicGlucoseDatabase</code> method), 137	<code>force_repair</code> ( <code>blatann.event_args.PeripheralSecurityRequestEventArgs</code> attribute), 156
<code>first_record()</code> ( <code>blatann.services.glucose.database.IGlucoseDatabase</code> method), 136	<code>force_repair()</code> ( <code>blatann.event_args.PeripheralSecurityRequestEventArgs</code> method), 156
<code>fitness_machine</code> ( <code>blatann.bt_sig.uuids.ServiceUuid</code> attribute), 32	<code>Format</code> (class in <code>blatann.bt_sig.assigned_numbers</code> ), 23
<code>fitness_machine_control_point</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39	<code>four_zone_heart_rate_limits</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39
<code>fitness_machine_feature</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39	<code>frequency_hertz</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 26
<code>fitness_machine_status</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39	<code>friday</code> ( <code>blatann.services.ble_data_types.DayOfWeek</code> attribute), 145
<code>five_zone_heart_rate_limits</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39	<code>from_array()</code> ( <code>blatann.nrf.nrf_types.generic.BLEUUID</code> class method), 114
<code>fixed_string_16</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39	<code>from_auth_request()</code> ( <code>blatann.nrf.nrf_events.gatt_events.GattsEvtRead</code> class method), 95
<code>fixed_string_24</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39	<code>from_auth_request()</code> ( <code>blatann.nrf.nrf_events.gatt_events.GattsEvtWrite</code> class method), 95
	<code>from_ble_adv_records()</code> ( <code>blatann.gap.advertise_data.AdvertisingData</code> class method), 62
	<code>from_buffer()</code> ( <code>blatann.gatt.SubscriptionState</code> class method), 77
	<code>from_c()</code> ( <code>blatann.nrf.nrf_events.gap_events.GapEvtAdvReport</code>

class method), 92  
 from\_c() (blatann.nrf.nrf\_events.gap\_events.GapEvtConnReq class method), 93  
 from\_c() (blatann.nrf.nrf\_events.gap\_events.GapEvtConnReq class method), 92  
 from\_c() (blatann.nrf.nrf\_events.gap\_events.GapEvtConnReq class method), 92  
 from\_c() (blatann.nrf.nrf\_events.gap\_events.GapEvtDataLenReq class method), 93  
 from\_c() (blatann.nrf.nrf\_events.gap\_events.GapEvtDataLenReq class method), 93  
 from\_c() (blatann.nrf.nrf\_events.gap\_events.GapEvtDisconnReq class method), 93  
 from\_c() (blatann.nrf.nrf\_events.gap\_events.GapEvtPhyUpReq class method), 93  
 from\_c() (blatann.nrf.nrf\_events.gap\_events.GapEvtPhyUpReq class method), 93  
 from\_c() (blatann.nrf.nrf\_events.gap\_events.GapEvtRssiChgReq class method), 92  
 from\_c() (blatann.nrf.nrf\_events.gap\_events.GapEvtTimeoReq class method), 92  
 from\_c() (blatann.nrf.nrf\_events.gatt\_events.GattcEvtAttrInfo128Req class method), 95  
 from\_c() (blatann.nrf.nrf\_events.gatt\_events.GattcEvtCharInfo128Req class method), 95  
 from\_c() (blatann.nrf.nrf\_events.gatt\_events.GattcEvtDescInfo128Req class method), 95  
 from\_c() (blatann.nrf.nrf\_events.gatt\_events.GattcEvtHvxReq class method), 94  
 from\_c() (blatann.nrf.nrf\_events.gatt\_events.GattcEvtMtuReq class method), 95  
 from\_c() (blatann.nrf.nrf\_events.gatt\_events.GattcEvtPrimReq class method), 94  
 from\_c() (blatann.nrf.nrf\_events.gatt\_events.GattcEvtReadReq class method), 94  
 from\_c() (blatann.nrf.nrf\_events.gatt\_events.GattcEvtTimeoReq class method), 95  
 from\_c() (blatann.nrf.nrf\_events.gatt\_events.GattcEvtWriteReq class method), 94  
 from\_c() (blatann.nrf.nrf\_events.gatt\_events.GattcEvtWriteReq class method), 94  
 from\_c() (blatann.nrf.nrf\_events.gatts\_evt\_exchange\_req class method), 96  
 from\_c() (blatann.nrf.nrf\_events.gatts\_evt\_handles\_req class method), 96  
 from\_c() (blatann.nrf.nrf\_events.gatts\_evt\_notify\_req class method), 96  
 from\_c() (blatann.nrf.nrf\_events.gatts\_evt\_read\_req class method), 96  
 from\_c() (blatann.nrf.nrf\_events.gatts\_evt\_sys\_attr\_req class method), 95  
 from\_c() (blatann.nrf.nrf\_events.gatts\_evt\_timeo\_req class method), 96  
 from\_c() (blatann.nrf.nrf\_events.gatts\_evt\_write\_req class method), 96  
 class method), 95  
 from\_c() (blatann.nrf.nrf\_events.generic\_events.EvtUserMemoryRequest class method), 96  
 from\_c() (blatann.nrf.nrf\_events.smp\_events.GapEvtAuthKeyRequest class method), 97  
 from\_c() (blatann.nrf.nrf\_events.smp\_events.GapEvtAuthStatus class method), 97  
 from\_c() (blatann.nrf.nrf\_events.smp\_events.GapEvtConnSecUpdate class method), 97  
 from\_c() (blatann.nrf.nrf\_events.smp\_events.GapEvtLescDhKeyRequest class method), 98  
 from\_c() (blatann.nrf.nrf\_events.smp\_events.GapEvtPasskeyDisplay class method), 97  
 from\_c() (blatann.nrf.nrf\_events.smp\_events.GapEvtSecInfoRequest class method), 97  
 from\_c() (blatann.nrf.nrf\_events.smp\_events.GapEvtSecParamsRequest class method), 97  
 from\_c() (blatann.nrf.nrf\_events.smp\_events.GapEvtSecRequest class method), 97  
 from\_c() (blatann.nrf.nrf\_types.gap.BLEAdvData class method), 110  
 from\_c() (blatann.nrf.nrf\_types.gap.BLEGapAddr class method), 108  
 from\_c() (blatann.nrf.nrf\_types.gap.BLEGapConnParams class method), 108  
 from\_c() (blatann.nrf.nrf\_types.gap.BLEGapPrivacyParams class method), 110  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattcAttrInfo128 class method), 112  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattcAttrInfo16 class method), 111  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattcDescriptor class method), 111  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattCharacteristic class method), 111  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattCharacteristicProperties class method), 111  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattcWriteParams class method), 111  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattExtendedCharacteristicProp class method), 111  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattsAttrMetadata class method), 112  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattsCharHandles class method), 112  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattsCharMetadata class method), 112  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattService class method), 111  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattsPresentationFormat class method), 112  
 from\_c() (blatann.nrf.nrf\_types.gatt.BLEGattsValue class method), 113  
 from\_c() (blatann.nrf.nrf\_types.generic.BLEUUID

- class method), 114
- from\_c() (blatann.nrf.nrf\_types.generic.BLEUUIDBase class method), 113
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapDhKey class method), 116
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapEncryptInfo class method), 115
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapEncryptKey class method), 115
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapIdKey class method), 115
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapMasterId class method), 115
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapPublicKey class method), 116
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapSecKeyDist class method), 114
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapSecKeys class method), 116
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapSecKeyset class method), 116
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapSecLevels class method), 114
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapSecMode class method), 114
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapSecParams class method), 114
- from\_c() (blatann.nrf.nrf\_types.smp.BLEGapSignKey class method), 116
- from\_dict() (blatann.gap.bond\_db.BondDbEntry class method), 66
- from\_dict() (blatann.gap.bond\_db.BondingData class method), 66
- from\_dict() (blatann.nrf.nrf\_types.smp.BLEGapEncryptInfo class method), 115
- from\_dict() (blatann.nrf.nrf\_types.smp.BLEGapEncryptKey class method), 115
- from\_dict() (blatann.nrf.nrf\_types.smp.BLEGapIdKey class method), 115
- from\_dict() (blatann.nrf.nrf\_types.smp.BLEGapMasterId class method), 115
- from\_dict() (blatann.nrf.nrf\_types.smp.BLEGapSignKey class method), 116
- from\_integer\_value() (blatann.services.ble\_data\_types.Bitfield class method), 145
- from\_keyset() (blatann.gap.bond\_db.BondingData class method), 66
- from\_notification\_complete\_event\_args() (blatann.event\_args.DecodedReadCompleteEventArgs static method), 157
- from\_read\_complete\_event\_args() (blatann.event\_args.DecodedReadCompleteEventArgs static method), 157
- from\_seconds() (blatann.services.current\_time.data\_types.DaylightSavingsTimeOffset static method), 123
- from\_string() (blatann.nrf.nrf\_types.gap.BLEGapAddr class method), 108
- from\_uuid128() (blatann.nrf.nrf\_types.generic.BLEUUID class method), 114
- from\_uuid128\_array() (blatann.nrf.nrf\_types.generic.BLEUUIDBase class method), 113
- front (blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor attribute), 24
- full\_hour\_dst (blatann.services.current\_time.data\_types.DaylightSavingsTimeOffset attribute), 123
- ## G
- gain\_settings\_attribute (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- gap\_auth\_payload\_timeout (blatann.nrf.nrf\_types.config.BleOptionFlag attribute), 98
- gap\_channel\_map (blatann.nrf.nrf\_types.config.BleOptionFlag attribute), 98
- gap\_compat\_mode\_1 (blatann.nrf.nrf\_types.config.BleOptionFlag attribute), 98
- gap\_local\_conn\_latency (blatann.nrf.nrf\_types.config.BleOptionFlag attribute), 98
- gap\_passkey (blatann.nrf.nrf\_types.config.BleOptionFlag attribute), 98
- gap\_scan\_req\_report (blatann.nrf.nrf\_types.config.BleOptionFlag attribute), 98
- gap\_slave\_latency\_disable (blatann.nrf.nrf\_types.config.BleOptionFlag attribute), 98
- GapEvt (class in blatann.nrf.nrf\_events.gap\_events), 92
- GapEvtAdvReport (class in blatann.nrf.nrf\_events.gap\_events), 92
- GapEvtAuthKeyRequest (class in blatann.nrf.nrf\_events.smp\_events), 97
- GapEvtAuthStatus (class in blatann.nrf.nrf\_events.smp\_events), 97
- GapEvtConnected (class in blatann.nrf.nrf\_events.gap\_events), 92
- GapEvtConnParamUpdate (class in blatann.nrf.nrf\_events.gap\_events), 92
- GapEvtConnParamUpdateRequest (class in blatann.nrf.nrf\_events.gap\_events), 92



GapEvtConnSecUpdate (class in <i>blatann.nrf.nrf_events.smp_events</i> ), 97	blatann.nrf.nrf_events.smp_events), 97	GattcOperationManager (class in <i>blatann.gatt.managers</i> ), 90	blatann.gatt.managers), 90
GapEvtDataLengthUpdate (class in <i>blatann.nrf.nrf_events.gap_events</i> ), 93	blatann.nrf.nrf_events.gap_events), 93	GattcReadCompleteEventArgs (class in <i>blatann.gatt.reader</i> ), 90	blatann.gatt.reader), 90
GapEvtDataLengthUpdateRequest (class in <i>blatann.nrf.nrf_events.gap_events</i> ), 93	blatann.nrf.nrf_events.gap_events), 93	GattcReader (class in <i>blatann.gatt.reader</i> ), 90	blatann.gatt.reader), 90
GapEvtDisconnected (class in <i>blatann.nrf.nrf_events.gap_events</i> ), 93	blatann.nrf.nrf_events.gap_events), 93	GattcService (class in <i>blatann.gatt.gattc</i> ), 81	blatann.gatt.gattc), 81
GapEvtLescDhKeyRequest (class in <i>blatann.nrf.nrf_events.smp_events</i> ), 97	blatann.nrf.nrf_events.smp_events), 97	GattcWriteCompleteEventArgs (class in <i>blatann.gatt.writer</i> ), 91	blatann.gatt.writer), 91
GapEvtPasskeyDisplay (class in <i>blatann.nrf.nrf_events.smp_events</i> ), 97	blatann.nrf.nrf_events.smp_events), 97	GattcWriter (class in <i>blatann.gatt.writer</i> ), 91	blatann.gatt.writer), 91
GapEvtPhyUpdate (class in <i>blatann.nrf.nrf_events.gap_events</i> ), 93	blatann.nrf.nrf_events.gap_events), 93	GattDatabase (class in <i>blatann.gatt</i> ), 77	blatann.gatt), 77
GapEvtPhyUpdateRequest (class in <i>blatann.nrf.nrf_events.gap_events</i> ), 93	blatann.nrf.nrf_events.gap_events), 93	GattEvt (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 94	blatann.nrf.nrf_events.gatt_events), 94
GapEvtRssiChanged (class in <i>blatann.nrf.nrf_events.gap_events</i> ), 92	blatann.nrf.nrf_events.gap_events), 92	GattOperationCompleteReason (class in <i>blatann.event_args</i> ), 154	blatann.event_args), 154
GapEvtSec (class in <i>blatann.nrf.nrf_events.smp_events</i> ), 97	blatann.nrf.nrf_events.smp_events), 97	GattsAttribute (class in <i>blatann.gatt.gatts_attribute</i> ), 88	blatann.gatt.gatts_attribute), 88
GapEvtSecInfoRequest (class in <i>blatann.nrf.nrf_events.smp_events</i> ), 97	blatann.nrf.nrf_events.smp_events), 97	GattsAttributeProperties (class in <i>blatann.gatt.gatts_attribute</i> ), 88	blatann.gatt.gatts_attribute), 88
GapEvtSecParamsRequest (class in <i>blatann.nrf.nrf_events.smp_events</i> ), 97	blatann.nrf.nrf_events.smp_events), 97	GattsCharacteristic (class in <i>blatann.gatt.gatts</i> ), 84	blatann.gatt.gatts), 84
GapEvtSecRequest (class in <i>blatann.nrf.nrf_events.smp_events</i> ), 97	blatann.nrf.nrf_events.smp_events), 97	GattsCharacteristicProperties (class in <i>blatann.gatt.gatts</i> ), 83	blatann.gatt.gatts), 83
GapEvtTimeout (class in <i>blatann.nrf.nrf_events.gap_events</i> ), 92	blatann.nrf.nrf_events.gap_events), 92	GattsDatabase (class in <i>blatann.gatt.gatts</i> ), 88	blatann.gatt.gatts), 88
GattcAttribute (class in <i>blatann.gatt.gattc_attribute</i> ), 83	blatann.gatt.gattc_attribute), 83	GattsEvt (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 94	blatann.nrf.nrf_events.gatt_events), 94
GattcCharacteristic (class in <i>blatann.gatt.gattc</i> ), 78	blatann.gatt.gattc), 78	GattsEvtExchangeMtuRequest (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 96	blatann.nrf.nrf_events.gatt_events), 96
GattcDatabase (class in <i>blatann.gatt.gattc</i> ), 82	blatann.gatt.gattc), 82	GattsEvtHandleValueConfirm (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 96	blatann.nrf.nrf_events.gatt_events), 96
GattcEvt (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 94	blatann.nrf.nrf_events.gatt_events), 94	GattsEvtNotificationTxComplete (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 96	blatann.nrf.nrf_events.gatt_events), 96
GattcEvtAttrInfoDiscoveryResponse (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 95	blatann.nrf.nrf_events.gatt_events), 95	GattsEvtRead (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 95	blatann.nrf.nrf_events.gatt_events), 95
GattcEvtCharacteristicDiscoveryResponse (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 94	blatann.nrf.nrf_events.gatt_events), 94	GattsEvtReadWriteAuthorizeRequest (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 95	blatann.nrf.nrf_events.gatt_events), 95
GattcEvtDescriptorDiscoveryResponse (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 95	blatann.nrf.nrf_events.gatt_events), 95	GattsEvtSysAttrMissing (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 95	blatann.nrf.nrf_events.gatt_events), 95
GattcEvtHvx (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 94	blatann.nrf.nrf_events.gatt_events), 94	GattsEvtTimeout (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 96	blatann.nrf.nrf_events.gatt_events), 96
GattcEvtMtuExchangeResponse (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 95	blatann.nrf.nrf_events.gatt_events), 95	GattsEvtWrite (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 95	blatann.nrf.nrf_events.gatt_events), 95
GattcEvtPrimaryServiceDiscoveryResponse (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 94	blatann.nrf.nrf_events.gatt_events), 94	GattsOperationManager (class in <i>blatann.gatt.managers</i> ), 90	blatann.gatt.managers), 90
GattcEvtReadResponse (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 94	blatann.nrf.nrf_events.gatt_events), 94	GattsService (class in <i>blatann.gatt.gatts</i> ), 87	blatann.gatt.gatts), 87
GattcEvtTimeout (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 95	blatann.nrf.nrf_events.gatt_events), 95	GattStatusCode (in module <i>blatann.gatt</i> ), 76	blatann.gatt), 76
GattcEvtWriteCmdTxComplete (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 94	blatann.nrf.nrf_events.gatt_events), 94	GattsUserDescriptionProperties (class in <i>blatann.gatt.gatts</i> ), 83	blatann.gatt.gatts), 83
GattcEvtWriteResponse (class in <i>blatann.nrf.nrf_events.gatt_events</i> ), 94	blatann.nrf.nrf_events.gatt_events), 94	gender ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 39	(blatann.bt_sig.uuids.CharacteristicUuid attribute), 39
		general_activity_instantaneous_data ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 39	(blatann.bt_sig.uuids.CharacteristicUuid attribute), 39
		general_activity_summary_data ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 39	(blatann.bt_sig.uuids.CharacteristicUuid attribute), 39
		general_device_fault ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 39	(blatann.bt_sig.uuids.CharacteristicUuid attribute), 39

<code>tann.services.glucose.data_types.GlucoseFeatureType</code> (attribute), 133	<code>get_firmware_revision()</code> (blatann.services.device_info.service.DisClient method), 129
<code>general_device_fault</code> (blatann.services.glucose.data_types.SensorStatusType attribute), 133	<code>get_gap_config()</code> (blatann.nrf.nrf_types.config.BleConnConfig method), 100
<code>GENERAL_DISCOVERY_MODE</code> (blatann.gap.advertise_data.AdvertisingFlags attribute), 59	<code>get_gatt_config()</code> (blatann.nrf.nrf_types.config.BleConnConfig method), 100
<code>generate_random_uuid128()</code> (in module blatann.uuid), 166	<code>get_gattc_config()</code> (blatann.nrf.nrf_types.config.BleConnConfig method), 100
<code>generate_random_uuid16()</code> (in module blatann.uuid), 165	<code>get_gatts_config()</code> (blatann.nrf.nrf_types.config.BleConnConfig method), 100
<code>generic_access</code> (blatann.bt_sig.uuids.ServiceUuid attribute), 32	<code>get_hardware_revision()</code> (blatann.services.device_info.service.DisClient method), 129
<code>generic_access_service</code> (blatann.device.BleDevice property), 153	<code>get_manufacturer_name()</code> (blatann.services.device_info.service.DisClient method), 129
<code>generic_attribute</code> (blatann.bt_sig.uuids.ServiceUuid attribute), 32	<code>get_model_number()</code> (blatann.services.device_info.service.DisClient method), 129
<code>generic_level</code> (blatann.bt_sig.uuids.CharacteristicUuid attribute), 39	<code>get_pnp_id()</code> (blatann.services.device_info.service.DisClient method), 129
<code>generic_media_control</code> (blatann.bt_sig.uuids.ServiceUuid attribute), 32	<code>get_records()</code> (blatann.services.glucose.database.BasicGlucoseDatabase method), 137
<code>generic_telephone_bearer</code> (blatann.bt_sig.uuids.ServiceUuid attribute), 33	<code>get_records()</code> (blatann.services.glucose.database.IGlucoseDatabase method), 136
<code>GenericAccessService</code> (class in blatann.gap.generic_access_service), 70	<code>get_regulatory_certifications()</code> (blatann.services.device_info.service.DisClient method), 129
<code>GenericWaitable</code> (class in blatann.waitables.waitable), 150	<code>get_report_for_peer()</code> (blatann.gap.advertise_data.ScanReportCollection method), 63
<code>get()</code> (blatann.services.device_info.service.DisClient method), 129	<code>get_role_count_cfg()</code> (blatann.nrf.nrf_types.config.BleEnableConfig method), 100
<code>get_addr_flag()</code> (blatann.nrf.nrf_types.gap.BLEGapAddr method), 109	<code>get_serial_number()</code> (blatann.services.device_info.service.DisClient method), 129
<code>get_addr_type_str()</code> (blatann.nrf.nrf_types.gap.BLEGapAddr method), 109	<code>get_service_changed_cfg()</code> (blatann.nrf.nrf_types.config.BleEnableConfig method), 100
<code>get_attr_tab_size_cfg()</code> (blatann.nrf.nrf_types.config.BleEnableConfig method), 100	<code>get_software_revision()</code> (blatann.services.device_info.service.DisClient method), 129
<code>get_configs()</code> (blatann.nrf.nrf_types.config.BleConnConfig method), 100	<code>get_system_id()</code> (blatann.services.device_info.service.DisClient method), 129
<code>get_configs()</code> (blatann.nrf.nrf_types.config.BleEnableConfig method), 100	<code>get_value()</code> (blatann.gatt.gatts_attribute.GattsAttribute method), 89
<code>get_device_name()</code> (blatann.nrf.nrf_events.gap_events.GapEvtAdvReport method), 92	<code>get_value()</code> (blatann.nrf.nrf_types.generic.BLEUUID method), 138
<code>get_device_name_cfg()</code> (blatann.nrf.nrf_types.config.BleEnableConfig method), 100	
<code>get_filter_min_max()</code> (blatann.services.glucose.racp.RacpCommand method), 138	

- method), 113
- get\_vs\_uuid\_cfg() (blatann.nrf.nrf\_types.config.BleEnableConfig method), 100
- global\_trade\_item\_number (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- glucose (blatann.bt\_sig.uuids.ServiceUuid attribute), 33
- glucose\_feature (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- glucose\_measurement (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- glucose\_measurement\_context (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- glucose\_meter (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 30
- GlucoseConcentrationUnits (class in blatann.services.glucose.data\_types), 130
- GlucoseContext (class in blatann.services.glucose.data\_types), 135
- GlucoseFeatures (class in blatann.services.glucose.data\_types), 133
- GlucoseFeatureType (class in blatann.services.glucose.data\_types), 133
- GlucoseMeasurement (class in blatann.services.glucose.data\_types), 134
- GlucoseSample (class in blatann.services.glucose.data\_types), 133
- GlucoseServer (class in blatann.services.glucose.service), 139
- GlucoseType (class in blatann.services.glucose.data\_types), 130
- gps (blatann.services.current\_time.data\_types.TimeSource attribute), 123
- greater\_than\_or\_equal\_to (blatann.services.glucose.racp.RacpOperator attribute), 138
- group\_object\_type (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- gust\_factor (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- ## H
- half\_hour\_dst (blatann.services.current\_time.data\_types.DaylightSavingTimeOffset attribute), 123
- handedness (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- handle (blatann.gatt.Attribute property), 77
- handle\_value\_array\_to\_list() (in module blatann.nrf.nrf\_driver\_types), 119
- hardware\_revision\_string (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- has\_handlers (blatann.event\_type.EventSource property), 158
- has\_local\_time\_info (blatann.services.current\_time.service.CurrentTimeClient property), 127
- has\_local\_time\_info (blatann.services.current\_time.service.CurrentTimeServer property), 125
- has\_reference\_info (blatann.services.current\_time.service.CurrentTimeClient property), 127
- has\_reference\_time\_info (blatann.services.current\_time.service.CurrentTimeServer property), 125
- HealthStatus (in module blatann.gap), 59
- health\_care\_professional (blatann.services.glucose.data\_types.TesterType attribute), 132
- health\_thermometer (blatann.bt\_sig.uuids.ServiceUuid attribute), 33
- HealthStatus (class in blatann.services.glucose.data\_types), 132
- hearing\_access (blatann.bt\_sig.uuids.ServiceUuid attribute), 33
- hearing\_aid\_features (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- hearing\_aid\_preset\_control\_point (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- heart\_rate (blatann.bt\_sig.uuids.ServiceUuid attribute), 33
- heart\_rate (blatann.nrf.nrf\_types.generic.BLEUUID.Standard attribute), 113
- heart\_rate\_control\_point (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- heart\_rate\_max (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- heart\_rate\_measurement (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39
- heart\_rate\_sensor (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 29
- heart\_rate\_sensor\_heart\_rate\_belt (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 29
- heat\_capacity\_joule\_per\_kelvin (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 39

<code>tann.bt_sig.assigned_numbers.Units</code> attribute), 26	<code>http_proxy</code> ( <code>blatann.bt_sig.uuids.ServiceUuid</code> attribute), 33
<code>heat_flux_density_watt_per_square_metre</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 26	<code>http_status_code</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>heat_index</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39	<code>https_security</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>height</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39	<code>human_interface_device</code> ( <code>blatann.bt_sig.uuids.ServiceUuid</code> attribute), 33
<code>HexConverterTest</code> (class in <code>blatann.examples.central_event_driven</code> ), 51	<code>humidity</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>hid</code> ( <code>blatann.bt_sig.assigned_numbers.Appearance</code> attribute), 30	<code>IdBasedEventWaitable</code> (class in <code>blatann.waitables.event_waitable</code> ), 148
<code>hid_barcode</code> ( <code>blatann.bt_sig.assigned_numbers.Appearance</code> attribute), 30	<code>idd_annunciation_status</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>hid_card_reader</code> ( <code>blatann.bt_sig.assigned_numbers.Appearance</code> attribute), 30	<code>idd_command_control_point</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>hid_control_point</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 39	<code>idd_command_data</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>hid_digital_pen</code> ( <code>blatann.bt_sig.assigned_numbers.Appearance</code> attribute), 30	<code>idd_features</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>hid_digitizer</code> ( <code>blatann.bt_sig.assigned_numbers.Appearance</code> attribute), 30	<code>idd_history_data</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>hid_gamepad</code> ( <code>blatann.bt_sig.assigned_numbers.Appearance</code> attribute), 30	<code>idd_record_access_control_point</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>hid_information</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40	<code>idd_status</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>hid_joystick</code> ( <code>blatann.bt_sig.assigned_numbers.Appearance</code> attribute), 30	<code>idd_status_changed</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>hid_keyboard</code> ( <code>blatann.bt_sig.assigned_numbers.Appearance</code> attribute), 30	<code>idd_status_reader_control_point</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>hid_mouse</code> ( <code>blatann.bt_sig.assigned_numbers.Appearance</code> attribute), 30	<code>ieee11073_20601_regulatory_certification_data_list</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>high_intensity_exercise_threshold</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40	<code>IGlucoseDatabase</code> (class in <code>blatann.services.glucose.database</code> ), 136
<code>high_resolution_height</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40	<code>illuminance</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40
<code>hip_circumference</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40	<code>illuminance_lux</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 26
<code>http_control_point</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40	<code>immediate_alert</code> ( <code>blatann.bt_sig.uuids.ServiceUuid</code> attribute), 33
<code>http_entity_body</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40	
<code>http_headers</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 40	



include (blatann.bt\_sig.uuids.DeclarationUuid attribute), 106  
 include\_array\_to\_list() (in module blatann.nrf.nrf\_driver\_types), 119  
 incoming\_call (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 40  
 incoming\_call\_target\_bearer\_uri (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 40  
 incorrect\_strip\_type (blatann.services.glucose.data\_types.SensorStatusType attribute), 132  
 INDICATION (blatann.gatt.SubscriptionState attribute), 77  
 indication (blatann.nrf.nrf\_types.enums.BLEGattHVXType attribute), 106  
 indoor\_bike\_data (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 40  
 indoor\_positioning (blatann.bt\_sig.uuids.ServiceUuid attribute), 33  
 indoor\_positioning\_configuration (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 40  
 inductance\_henry (blatann.bt\_sig.assigned\_numbers.Units attribute), 26  
 information\_3d\_data (blatann.gap.advertise\_data.AdvertisingData.Types attribute), 61  
 information\_3d\_data (blatann.nrf.nrf\_types.gap.BLEAdvData.Types attribute), 110  
 initialize() (blatann.services.nordic\_uart.service.NordicUartClient attribute), 141  
 inside (blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor attribute), 24  
 instant\_passed (blatann.nrf.nrf\_types.enums.BLEHci attribute), 101  
 insuf\_authentication (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106  
 insuf\_authorization (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106  
 insuf\_enc\_key\_size (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106  
 insuf\_encryption (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106  
 insuf\_resources (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106  
 insulin\_delivery (blatann.bt\_sig.uuids.ServiceUuid attribute), 33  
 Int16 (class in blatann.services.ble\_data\_types), 142  
 Int32 (class in blatann.services.ble\_data\_types), 143  
 Int64 (class in blatann.services.ble\_data\_types), 143  
 Int8 (class in blatann.services.ble\_data\_types), 142  
 IntEnumWithDescription (class in blatann.utils), 146  
 intermediate\_acting\_insulin (blatann.services.glucose.data\_types.MedicationType attribute), 132  
 intermediate\_cuff\_pressure (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 40  
 intermediate\_temperature (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 40  
 internal (blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor attribute), 24  
 internal (blatann.nrf.nrf\_types.enums.NrfError attribute), 102  
 internet\_protocol\_support (blatann.bt\_sig.uuids.ServiceUuid attribute), 33  
 interstitial\_fluid (blatann.services.glucose.data\_types.GlucoseType attribute), 130  
 interval\_ms (blatann.gap.gap\_types.ActiveConnectionParameters property), 70  
 invalid (blatann.nrf.nrf\_types.enums.BLEGapRoles attribute), 104  
 invalid (blatann.nrf.nrf\_types.enums.BLEGattHVXType attribute), 106  
 invalid (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106  
 invalid (blatann.nrf.nrf\_types.enums.BLEGattsWriteOperation attribute), 107  
 invalid (blatann.nrf.nrf\_types.enums.BLEGattWriteOperation attribute), 105  
 invalid\_addr (blatann.nrf.nrf\_types.enums.NrfError attribute), 102  
 invalid\_att\_val\_length (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106  
 invalid\_btble\_command\_parameters (blatann.nrf.nrf\_types.enums.BLEHci attribute), 101  
 invalid\_data (blatann.nrf.nrf\_types.enums.NrfError attribute), 102  
 invalid\_flags (blatann.nrf.nrf\_types.enums.NrfError attribute), 102  
 invalid\_handle (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106

`invalid_length` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 102

`invalid_lmp_parameters` (`blatann.nrf.nrf_types.enums.BLEHci` attribute), 101

`invalid_offset` (`blatann.nrf.nrf_types.enums.BLEGattStatusCode` attribute), 106

`invalid_operand` (`blatann.services.glucose.racp.RacpResponseCode` attribute), 138

`invalid_operator` (`blatann.services.glucose.racp.RacpResponseCode` attribute), 138

`invalid_param` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 102

`invalid_params` (`blatann.nrf.nrf_types.enums.BLEGapSecStatus` attribute), 105

`invalid_pdu` (`blatann.nrf.nrf_types.enums.BLEGattStatusCode` attribute), 106

`invalid_state` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 102

`InvalidOperationException`, 159

`InvalidStateException`, 159

`irradiance` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 40

`irradiance_watt_per_square_metre` (`blatann.bt_sig.assigned_numbers.Units` attribute), 26

`is_advertising` (`blatann.gap.advertising.Advertiser` property), 64

`is_bonded_device` (`blatann.gap.advertise_data.ScanReport` property), 62

`is_client` (`blatann.peer.Peer` property), 160

`is_in_range()` (`blatann.nrf.nrf_types.gap.TimeRange` method), 108

`is_initialized` (`blatann.services.nordic_uart.service.NordicUartClient` property), 141

`is_open` (`blatann.nrf.nrf_driver.NrfDriver` property), 117

`is_peripheral` (`blatann.peer.Peer` property), 160

`is_previously_bonded` (`blatann.gap.smp.SecurityManager` property), 74

`is_previously_bonded` (`blatann.peer.Peer` property), 160

`is_running` (`blatann.utils.Stopwatch` property), 146

`is_scanning` (`blatann.gap.scanning.Scanner` property), 71

`is_valid` (`blatann.nrf.nrf_types.smp.BLEGapMasterId` property), 115

`is_writable` (`blatann.services.current_time.service.CurrentTimeServer` property), 125

`iter_characteristics()` (`blatann.gatt.gattc.GattcDatabase` method), 82

`iter_services()` (`blatann.gatt.gatts.GattsDatabase` method), 88

## J

`join()` (`blatann.examples.peripheral.CountingCharacteristicThread` method), 54

`JsonDatabaseStrategy` (class in `blatann.gap.default_bond_db`), 67

`JUST_WORKS` (`blatann.gap.smp.SecurityLevel` attribute), 72

## K

`KEY_LENGTH` (`blatann.nrf.nrf_types.smp.BLEGapDhKey` attribute), 116

`KEY_LENGTH` (`blatann.nrf.nrf_types.smp.BLEGapEncryptInfo` attribute), 115

`KEY_LENGTH` (`blatann.nrf.nrf_types.smp.BLEGapIdKey` attribute), 115

`KEY_LENGTH` (`blatann.nrf.nrf_types.smp.BLEGapPublicKey` attribute), 115

`KEY_LENGTH` (`blatann.nrf.nrf_types.smp.BLEGapSignKey` attribute), 116

`KEYBOARD_DISPLAY` (`blatann.nrf.nrf_types.enums.BLEGapIoCaps` attribute), 104

`KEYBOARD_ONLY` (`blatann.nrf.nrf_types.enums.BLEGapIoCaps` attribute), 104

`keyring` (`blatann.bt_sig.assigned_numbers.Appearance` attribute), 29

`kg_per_liter` (`blatann.services.glucose.data_types.GlucoseConcentration` attribute), 130

## L

`lab_test` (`blatann.services.glucose.data_types.TesterType` attribute), 132

`language` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 40

`last_name` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 40

`last_record` (`blatann.services.glucose.racp.RacpOperator` attribute), 138

`last_record()` (`blatann.services.glucose.database.BasicGlucoseDatabase` method), 137

`last_record()` (`blatann.services.glucose.database.IGlucoseDatabase` method), 136

`latitude` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 40

<code>le_bluetooth_device_address</code> ( <i>blatann.gap.advertise_data.AdvertisingData.Types</i> attribute), 61	<code>tann.nrf.nrf_driver_types</code> ), 120
<code>le_bluetooth_device_address</code> ( <i>blatann.nrf.nrf_types.gap.BLEAdvData.Types</i> attribute), 109	<code>list_to_char_array()</code> (in module <i>blatann.nrf.nrf_driver_types</i> ), 120
<code>le_role</code> ( <i>blatann.gap.advertise_data.AdvertisingData.Types</i> attribute), 61	<code>list_to_desc_array()</code> (in module <i>blatann.nrf.nrf_driver_types</i> ), 120
<code>le_role</code> ( <i>blatann.nrf.nrf_types.gap.BLEAdvData.Types</i> attribute), 110	<code>list_to_handle_value_array()</code> (in module <i>blatann.nrf.nrf_driver_types</i> ), 120
<code>left</code> ( <i>blatann.bt_sig.assigned_numbers.NamespaceDescriptor</i> attribute), 24	<code>list_to_include_array()</code> (in module <i>blatann.nrf.nrf_driver_types</i> ), 120
<code>length_angstrom</code> ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 27	<code>list_to_serial_port_desc_array()</code> (in module <i>blatann.nrf.nrf_driver_types</i> ), 120
<code>length_foot</code> ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 26	<code>list_to_service_array()</code> (in module <i>blatann.nrf.nrf_driver_types</i> ), 120
<code>length_inch</code> ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 26	<code>list_to_uint16_array()</code> (in module <i>blatann.nrf.nrf_driver_types</i> ), 120
<code>length_metre</code> ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 26	<code>list_to_uint8_array()</code> (in module <i>blatann.nrf.nrf_driver_types</i> ), 120
<code>length_mile</code> ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 26	<code>lmp_pdu_not_allowed</code> ( <i>blatann.nrf.nrf_types.enums.BLEHci</i> attribute), 101
<code>length_nautical_mile</code> ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 26	<code>lmp_response_timeout</code> ( <i>blatann.nrf.nrf_types.enums.BLEHci</i> attribute), 101
<code>length_parsec</code> ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 26	<code>lmp_transaction_collision</code> ( <i>blatann.nrf.nrf_types.enums.BLEHci</i> attribute), 101
<code>length_yard</code> ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 27	<code>ln_control_point</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 41
<code>lesc_compute_dh_key()</code> (in module <i>blatann.gap.smp_crypto</i> ), 75	<code>ln_feature</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 41
<code>lesc_generate_private_key()</code> (in module <i>blatann.gap.smp_crypto</i> ), 75	<code>load()</code> ( <i>blatann.gap.bond_db.BondDatabaseLoader</i> method), 66
<code>LESC_MITM</code> ( <i>blatann.gap.smp.SecurityLevel</i> attribute), 72	<code>load()</code> ( <i>blatann.gap.default_bond_db.DatabaseStrategy</i> method), 67
<code>LESC_MITM</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapSecModeType</i> attribute), 114	<code>load()</code> ( <i>blatann.gap.default_bond_db.DefaultBondDatabaseLoader</i> method), 68
<code>lesc_privkey_from_raw()</code> (in module <i>blatann.gap.smp_crypto</i> ), 75	<code>load()</code> ( <i>blatann.gap.default_bond_db.JsonDatabaseStrategy</i> method), 67
<code>lesc_privkey_to_raw()</code> (in module <i>blatann.gap.smp_crypto</i> ), 75	<code>load()</code> ( <i>blatann.gap.default_bond_db.PickleDatabaseStrategy</i> method), 68
<code>lesc_pubkey_from_raw()</code> (in module <i>blatann.gap.smp_crypto</i> ), 75	<code>local_east_coordinate</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 41
<code>lesc_pubkey_to_raw()</code> (in module <i>blatann.gap.smp_crypto</i> ), 75	<code>local_host_terminated_connection</code> ( <i>blatann.nrf.nrf_types.enums.BLEHci</i> attribute), 101
<code>less_than_or_equal_to</code> ( <i>blatann.services.glucose.racp.RacpOperator</i> attribute), 138	<code>local_north_coordinate</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 41
<code>LIMITED_DISCOVERY_MODE</code> ( <i>blatann.gap.advertise_data.AdvertisingFlags</i> attribute), 59	<code>local_time_information</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 41
<code>link_loss</code> ( <i>blatann.bt_sig.uuids.ServiceUuid</i> attribute), 33	<code>LocalTimeInfo</code> (class in <i>bla-</i>
<code>list_to_ble_gattc_char_array()</code> (in module <i>bla-</i>	

- tann.services.current\_time.data\_types*), 124
- location\_and\_navigation** (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 33
- location\_and\_speed** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- location\_name** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- logarithmic\_radio\_quantity\_bel** (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27
- logarithmic\_radio\_quantity\_neper** (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27
- logger** (in module *blatann.gatt*), 76
- long\_acting\_insulin** (*blatann.services.glucose.data\_types.MedicationType* attribute), 132
- longitude** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- low\_battery\_detection** (*blatann.services.glucose.data\_types.GlucoseFeatureType* attribute), 133
- lower** (*blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor* attribute), 25
- luminance\_candela\_per\_square\_metre** (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27
- luminous\_efficacy** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- luminous\_efficacy\_lumen\_per\_watt** (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27
- luminous\_energy** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- luminous\_energy\_lumen\_hour** (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27
- luminous\_exposure** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- luminous\_exposure\_lux\_hour** (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27
- luminous\_flux** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- luminous\_flux\_lumen** (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27
- luminous\_flux\_range** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- luminous\_intensity** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- luminous\_intensity\_candela** (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27
- lunch** (*blatann.services.glucose.data\_types.CarbohydrateType* attribute), 131
- ## M
- magnetic\_declination** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- magnetic\_field\_strength\_ampere\_per\_metre** (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27
- magnetic\_flux\_density\_2d** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- magnetic\_flux\_density\_3d** (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 41
- magnetic\_flux\_density\_tesla** (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27
- magnetic\_flux\_weber** (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27
- main** (*blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor* attribute), 25
- main()** (in module *blatann.examples.broadcaster*), 49
- main()** (in module *blatann.examples.central\_uart\_service*), 50
- main()** (in module *blatann.examples.central*), 50
- main()** (in module *blatann.examples.central\_battery\_service*), 51
- main()** (in module *blatann.examples.central\_descriptors*), 51
- main()** (in module *blatann.examples.central\_device\_info\_service*), 51
- main()** (in module *blatann.examples.central\_event\_driven*), 52
- main()** (in module *blatann.examples.peripheral*), 54
- main()** (in module *blatann.examples.peripheral\_battery\_service*), 55
- main()** (in module *blatann.examples.peripheral\_current\_time\_service*), 55
- main()** (in module *blatann.examples.peripheral\_descriptors*), 56

`main()` (in module `blatann.examples.peripheral_device_info_service`), 56  
`main()` (in module `blatann.examples.peripheral_glucose_service`), 57  
`main()` (in module `blatann.examples.peripheral_rssi`), 58  
`main()` (in module `blatann.examples.peripheral_uart_service`), 59  
`main()` (in module `blatann.examples.scanner`), 59  
`major_issues` (`blatann.services.glucose.data_types.HealthStatus` property), 132  
`manual` (`blatann.services.current_time.data_types.TimeSource` attribute), 123  
`manual_time_update` (`blatann.services.current_time.data_types.AdjustmentReasonType` attribute), 123  
`manufacturer_data` (`blatann.gap.advertise_data.AdvertisingData` property), 61  
`manufacturer_name_string` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 41  
`manufacturer_specific_data` (`blatann.gap.advertise_data.AdvertisingData.Types` attribute), 61  
`manufacturer_specific_data` (`blatann.nrf.nrf_types.gap.BLEAdvData.Types` attribute), 110  
`mark()` (`blatann.utils.Stopwatch` method), 146  
`mass_concentration_kilogram_per_cubic_metre` (`blatann.bt_sig.assigned_numbers.Units` attribute), 27  
`mass_density_milligram_per_decilitre` (`blatann.bt_sig.assigned_numbers.Units` attribute), 27  
`mass_density_millimole_per_litre` (`blatann.bt_sig.assigned_numbers.Units` attribute), 27  
`mass_flow` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 41  
`mass_flow_gram_per_second` (`blatann.bt_sig.assigned_numbers.Units` attribute), 27  
`mass_kilogram` (`blatann.bt_sig.assigned_numbers.Units` attribute), 27  
`mass_pound` (`blatann.bt_sig.assigned_numbers.Units` attribute), 27  
`mass_tonne` (`blatann.bt_sig.assigned_numbers.Units` attribute), 27  
`match_confirm()` (`blatann.event_args.PasskeyDisplayEventArgs` method), 155  
`matches_peer()` (`blatann.gap.bond_db.BondDbEntry` method), 66  
`max` (`blatann.nrf.nrf_types.gap.TimeRange` property), 108  
`MAX_ENCODED_LENGTH` (`blatann.gap.advertise_data.AdvertisingData` attribute), 60  
`max_interval_ms` (`blatann.gap.advertising.Advertiser` property), 64  
`max_length` (`blatann.gatt.gatts.GattsCharacteristic` property), 86  
`max_length` (`blatann.gatt.gatts_attribute.GattsAttribute` property), 89  
`max_mtu_size` (`blatann.device.BleDevice` property), 153  
`max_mtu_size` (`blatann.peer.Peer` property), 160  
`max_write_length` (`blatann.services.nordic_uart.service.NordicUartClient` property), 140  
`max_write_length` (`blatann.services.nordic_uart.service.NordicUartServer` property), 140  
`maximum_recommended_heart_rate` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 41  
`MealType` (class in `blatann.services.glucose.data_types`), 131  
`measurement_interval` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 41  
`media_control` (`blatann.bt_sig.uuids.ServiceUuid` attribute), 33  
`media_control_point` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 41  
`media_control_point_opcodes_supported` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 41  
`media_player` (`blatann.bt_sig.assigned_numbers.Appearance` attribute), 29  
`media_player_icon_object_id` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 41  
`media_player_icon_object_type` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 41  
`media_player_icon_url` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 41  
`media_player_name` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 41  
`media_state` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 41  
`MedicationInfo` (class in `blatann.services.glucose.data_types`), 135



MedicationType	(class in <i>blatann.services.glucose.data_types</i> ), 132	model_number_string	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42
MedicationUnits	(class in <i>blatann.services.glucose.data_types</i> ), 131	module	
memory_capacity_exceeded	( <i>blatann.nrf.nrf_types.enums.BLEHci</i> attribute), 101	blatann,	23
mesh_provisioning	( <i>blatann.bt_sig.uuids.ServiceUuid</i> attribute), 33	blatann.bt_sig,	23
mesh_provisioning_data_in	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 41	blatann.bt_sig.assigned_numbers,	23
mesh_provisioning_data_out	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 41	blatann.bt_sig.uuids,	31
mesh_proxy	( <i>blatann.bt_sig.uuids.ServiceUuid</i> attribute), 33	blatann.device,	151
mesh_proxy_data_in	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 41	blatann.event_args,	154
mesh_proxy_data_out	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42	blatann.event_type,	158
metabolic_equivalent	( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 27	blatann.examples,	49
methane_concentration	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42	blatann.examples.broadcaster,	49
microphone_control	( <i>blatann.bt_sig.uuids.ServiceUuid</i> attribute), 33	blatann.examples.central_uart_service,	49
middle_name	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42	blatann.examples.central,	50
migrate_bond_database()	(in module <i>blatann.gap.default_bond_db</i> ), 69	blatann.examples.central_battery_service,	51
migrate_to_json()	( <i>blatann.gap.default_bond_db.DefaultBondDatabaseLoader</i> method), 68	blatann.examples.central_descriptors,	51
milligrams	( <i>blatann.services.glucose.data_types.MedicationUnits</i> attribute), 131	blatann.examples.central_device_info_service,	51
milliliters	( <i>blatann.services.glucose.data_types.MedicationUnits</i> attribute), 131	blatann.examples.central_event_driven,	51
min	( <i>blatann.nrf.nrf_types.gap.TimeRange</i> property), 108	blatann.examples.constants,	52
min_interval_ms	( <i>blatann.gap.advertising.Advertiser</i> property), 64	blatann.examples.example_utils,	52
minor_issues	( <i>blatann.services.glucose.data_types.HealthStatus</i> attribute), 132	blatann.examples.peripheral,	52
missing_handles()	( <i>blatann.nrf.nrf_types.gatt.BLEGattCharacteristic</i> method), 111	blatann.examples.peripheral_battery_service,	54
MITM	( <i>blatann.gap.smp.SecurityLevel</i> attribute), 72	blatann.examples.peripheral_current_time_service,	55
MITM	( <i>blatann.nrf.nrf_types.smp.BLEGapSecModeType</i> attribute), 114	blatann.examples.peripheral_descriptors,	56
		blatann.examples.peripheral_device_info_service,	56
		blatann.examples.peripheral_glucose_service,	57
		blatann.examples.peripheral_rssi,	58
		blatann.examples.peripheral_uart_service,	58
		blatann.examples.scanner,	59
		blatann.exceptions,	159
		blatann.gap,	59
		blatann.gap.advertise_data,	59
		blatann.gap.advertising,	64
		blatann.gap.bond_db,	66
		blatann.gap.default_bond_db,	67
		blatann.gap.gap_types,	69
		blatann.gap.generic_access_service,	70
		blatann.gap.scanning,	71
		blatann.gap.smp,	72
		blatann.gap.smp_crypto,	75
		blatann.gatt,	76
		blatann.gatt.gattc,	78
		blatann.gatt.gattc_attribute,	83
		blatann.gatt.gatts,	83

blatann.gatt.gatts\_attribute, 88  
 blatann.gatt.managers, 90  
 blatann.gatt.reader, 90  
 blatann.gatt.service\_discovery, 91  
 blatann.gatt.writer, 91  
 blatann.nrf, 92  
 blatann.nrf.nrf\_dll\_load, 116  
 blatann.nrf.nrf\_driver, 116  
 blatann.nrf.nrf\_driver\_types, 119  
 blatann.nrf.nrf\_events, 92  
 blatann.nrf.nrf\_events.gap\_events, 92  
 blatann.nrf.nrf\_events.gatt\_events, 94  
 blatann.nrf.nrf\_events.generic\_events, 96  
 blatann.nrf.nrf\_events.smp\_events, 97  
 blatann.nrf.nrf\_types, 98  
 blatann.nrf.nrf\_types.config, 98  
 blatann.nrf.nrf\_types.enums, 100  
 blatann.nrf.nrf\_types.gap, 108  
 blatann.nrf.nrf\_types.gatt, 110  
 blatann.nrf.nrf\_types.generic, 113  
 blatann.nrf.nrf\_types.smp, 114  
 blatann.peer, 159  
 blatann.services, 120  
 blatann.services.battery, 120  
 blatann.services.battery.constants, 121  
 blatann.services.battery.data\_types, 121  
 blatann.services.battery.service, 121  
 blatann.services.ble\_data\_types, 141  
 blatann.services.current\_time, 122  
 blatann.services.current\_time.constants, 123  
 blatann.services.current\_time.data\_types, 123  
 blatann.services.current\_time.service, 125  
 blatann.services.decoded\_event\_dispatcher, 146  
 blatann.services.device\_info, 127  
 blatann.services.device\_info.constants, 128  
 blatann.services.device\_info.data\_types, 128  
 blatann.services.device\_info.service, 129  
 blatann.services.glucose, 130  
 blatann.services.glucose.constants, 130  
 blatann.services.glucose.data\_types, 130  
 blatann.services.glucose.database, 136  
 blatann.services.glucose.racp, 137  
 blatann.services.glucose.service, 139  
 blatann.services.nordic\_uart, 139  
 blatann.services.nordic\_uart.constants, 140  
 blatann.services.nordic\_uart.service, 140  
 blatann.utils, 146  
 blatann.utils.queued\_tasks\_manager, 147  
 blatann.uuid, 165  
 blatann.waitables, 147  
 blatann.waitables.connection\_waitable, 147  
 blatann.waitables.event\_waitable, 148  
 blatann.waitables.scan\_waitable, 149  
 blatann.waitables.waitable, 149  
 mol\_per\_liter (*blatann.services.glucose.data\_types.GlucoseConcentration* attribute), 130  
 molar\_energy\_joule\_per\_mole (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27  
 molar\_entropy\_joule\_per\_mole\_kelvin (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27  
 moment\_of\_force\_newton\_metre (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 27  
 monday (*blatann.services.ble\_data\_types.DayOfWeek* attribute), 144  
 msec\_to\_units() (in module *blatann.nrf.nrf\_driver\_types*), 119  
 mtu\_size (*blatann.peer.Peer* property), 160  
 MTU\_SIZE\_DEFAULT (in module *blatann.gatt*), 76  
 MTU\_SIZE\_MINIMUM (in module *blatann.gatt*), 76  
 MtuSizeUpdatedEventArgs (class in *blatann.event\_args*), 155  
 multiple\_bond (*blatann.services.glucose.data\_types.GlucoseFeatureType* attribute), 133  
 mute (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 42  
 MyPeripheralConnection (class in *blatann.examples.central\_event\_driven*), 52

## N

name (*blatann.nrf.nrf\_types.gap.TimeRange* property), 108  
 name (*blatann.peer.Peer* property), 159  
 Namespace (class in *blatann.bt\_sig.assigned\_numbers*), 24  
 NamespaceDescriptor (class in *blatann.bt\_sig.assigned\_numbers*), 24  
 NAN (*blatann.services.ble\_data\_types.SFloat.ReservedMantissaValues* attribute), 144  
 navigation (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 42  
 NEG\_INFINITY (*blatann.services.ble\_data\_types.SFloat.ReservedMantissaValues* attribute), 144  
 network\_availability (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 42  
 network\_time\_protocol (*blatann.services.current\_time.data\_types.TimeSource*

- attribute), 123
- new\_alert (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 42
- new\_uuid\_from\_base() (blatann.uuid.Uuid128 method), 165
- next() (blatann.utils.SynchronousMonotonicCounter method), 146
- next\_dst\_change (blatann.bt\_sig.uuids.ServiceUuid attribute), 33
- next\_track\_object\_id (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 42
- nibble (blatann.bt\_sig.assigned\_numbers.Format attribute), 23
- nitrogen\_dioxide\_concentration (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 42
- NO\_ACCESS (blatann.gap.smp.SecurityLevel attribute), 72
- NO\_ACCESS (blatann.nrf.nrf\_types.smp.BLEGapSecModeType attribute), 114
- no\_mem (blatann.nrf.nrf\_types.enums.NrfError attribute), 102
- no\_records\_found (blatann.services.glucose.racp.RacpResponseCode attribute), 138
- non\_bonded\_central\_request (blatann.event\_args.PairingRejectedReason attribute), 156
- non\_bonded\_peripheral\_request (blatann.event\_args.PairingRejectedReason attribute), 156
- non\_connectable\_undirected (blatann.nrf.nrf\_types.enums.BLEGapAdvType attribute), 104
- non\_methane\_volatile\_organic\_compounds\_concentration (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 42
- NONE (blatann.nrf.nrf\_types.enums.BLEGapAuthKeyType attribute), 105
- NONE (blatann.nrf.nrf\_types.enums.BLEGapIoCaps attribute), 104
- NordicSemiErrorCheck() (in module blatann.nrf.nrf\_driver), 116
- NordicUartClient (class in blatann.services.nordic\_uart.service), 140
- NordicUartServer (class in blatann.services.nordic\_uart.service), 140
- normal (blatann.services.glucose.data\_types.HealthStatus attribute), 132
- not\_available (blatann.services.glucose.data\_types.HealthStatus attribute), 132
- not\_available (blatann.services.glucose.data\_types.TesterType attribute), 132
- not\_found (blatann.nrf.nrf\_types.enums.NrfError attribute), 102
- NOT\_SUBSCRIBED (blatann.gatt.SubscriptionState attribute), 77
- not\_supported (blatann.nrf.nrf\_types.enums.NrfError attribute), 102
- not\_supported (blatann.services.glucose.racp.RacpResponseCode attribute), 138
- notifiable (blatann.gatt.gatts.GattsCharacteristic property), 86
- notification (blatann.nrf.nrf\_types.enums.BLEGattHVXType attribute), 106
- NOTIFICATION\_INDICATION\_OVERHEAD\_BYTES (blatann.peer.Peer attribute), 159
- NotificationCompleteEventArgs (class in blatann.event\_args), 157
- NotificationReceivedEventArgs (class in blatann.event\_args), 157
- NOTIFY (blatann.gatt.SubscriptionState attribute), 77
- notify() (blatann.event\_type.EventSource method), 158
- notify() (blatann.gatt.gatts.GattsCharacteristic method), 84
- notify() (blatann.gatt.managers.GattsOperationManager method), 90
- notify() (blatann.waitables.waitable.GenericWaitable method), 150
- NRES (blatann.services.ble\_data\_types.SFloat.ReservedMantissaValues attribute), 144
- NrfDriver (class in blatann.nrf.nrf\_driver), 116
- NrfDriverObserver (class in blatann.nrf.nrf\_driver), 116
- NrfError (class in blatann.nrf.nrf\_types.enums), 101
- null (blatann.nrf.nrf\_types.enums.NrfError attribute), 102
- null (blatann.services.glucose.racp.RacpOperator attribute), 138
- num\_comp\_failure (blatann.nrf.nrf\_types.enums.BLEGapSecStatus attribute), 105
- number\_of\_digitals (blatann.bt\_sig.uuids.DescriptorUuid attribute), 31
- number\_of\_records\_response (blatann.services.glucose.racp.RacpOpcode attribute), 137
- ## O
- object\_action\_control\_point (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 42
- object\_changed (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 42
- object\_first\_created (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 42



- tribute*), 42
- `object_id` (*blatann.bt\_sig.uuids.CharacteristicUuid attribute*), 42
- `object_last_modified` (*blatann.bt\_sig.uuids.CharacteristicUuid attribute*), 42
- `object_list_control_point` (*blatann.bt\_sig.uuids.CharacteristicUuid attribute*), 42
- `object_list_filter` (*blatann.bt\_sig.uuids.CharacteristicUuid attribute*), 42
- `object_name` (*blatann.bt\_sig.uuids.CharacteristicUuid attribute*), 42
- `object_properties` (*blatann.bt\_sig.uuids.CharacteristicUuid attribute*), 42
- `object_size` (*blatann.bt\_sig.uuids.CharacteristicUuid attribute*), 42
- `object_transfer` (*blatann.bt\_sig.uuids.ServiceUuid attribute*), 33
- `object_type` (*blatann.bt\_sig.uuids.CharacteristicUuid attribute*), 42
- `observer_register()` (*blatann.nrf.nrf\_driver.NrfDriver method*), 117
- `observer_unregister()` (*blatann.nrf.nrf\_driver.NrfDriver method*), 117
- `on_advertising_timeout` (*blatann.gap.advertising.Advertiser property*), 64
- `on_battery_level_update()` (*in module blatann.examples.central\_battery\_service*), 51
- `on_battery_level_updated` (*blatann.services.battery.service.BatteryClient property*), 121
- `on_client_pairing_complete()` (*in module blatann.examples.peripheral*), 54
- `on_conn_params_updated()` (*in module blatann.examples.peripheral*), 54
- `on_connect` (*blatann.peer.Peer property*), 161
- `on_connect()` (*in module blatann.examples.central\_uart\_service*), 49
- `on_connect()` (*in module blatann.examples.peripheral*), 52
- `on_connect()` (*in module blatann.examples.peripheral\_battery\_service*), 54
- `on_connect()` (*in module blatann.examples.peripheral\_current\_time\_service*), 55
- `on_connect()` (*in module blatann.examples.peripheral\_descriptors*), 56
- `on_connect()` (*in module blatann.examples.peripheral\_device\_info\_service*), 56
- `on_connect()` (*in module blatann.examples.peripheral\_glucose\_service*), 57
- `on_connect()` (*in module blatann.examples.peripheral\_rssi*), 58
- `on_connect()` (*in module blatann.examples.peripheral\_uart\_service*), 58
- `on_connection_parameters_updated` (*blatann.peer.Peer property*), 161
- `on_counting_char_notification()` (*in module blatann.examples.central*), 50
- `on_current_time_updated` (*blatann.services.current\_time.service.CurrentTimeClient property*), 127
- `on_current_time_write` (*blatann.services.current\_time.service.CurrentTimeServer property*), 126
- `on_current_time_write()` (*in module blatann.examples.peripheral\_current\_time\_service*), 55
- `on_data_length_updated` (*blatann.peer.Peer property*), 161
- `on_data_received` (*blatann.services.nordic\_uart.service.NordicUartClient property*), 140
- `on_data_received` (*blatann.services.nordic\_uart.service.NordicUartServer property*), 140
- `on_data_rx()` (*in module blatann.examples.central\_uart\_service*), 49
- `on_data_rx()` (*in module blatann.examples.peripheral\_uart\_service*), 59
- `on_database_discovery_complete` (*blatann.peer.Peer property*), 162
- `on_disconnect` (*blatann.peer.Peer property*), 161
- `on_disconnect()` (*in module blatann.examples.central\_uart\_service*), 49
- `on_disconnect()` (*in module blatann.examples.peripheral*), 52
- `on_disconnect()` (*in module blatann.examples.peripheral\_battery\_service*), 54
- `on_disconnect()` (*in module blatann.examples.peripheral\_current\_time\_service*), 55
- `on_disconnect()` (*in module blatann.examples.peripheral\_descriptors*), 56
- `on_disconnect()` (*in module blatann.examples.peripheral\_device\_info\_service*), 56

<code>on_disconnect()</code> (in module <code>blatann.examples.peripheral_glucose_service</code> ), 57	<code>on_passkey_entry()</code> (in module <code>blatann.examples.central</code> ), 50
<code>on_disconnect()</code> (in module <code>blatann.examples.peripheral_rssi</code> ), 58	<code>on_passkey_entry()</code> (in module <code>blatann.examples.peripheral</code> ), 54
<code>on_disconnect()</code> (in module <code>blatann.examples.peripheral_uart_service</code> ), 58	<code>on_passkey_required</code> ( <code>blatann.gap.smp.SecurityManager</code> property), 73
<code>on_discovery_complete</code> ( <code>blatann.gatt.service_discovery.DatabaseDiscoverer</code> property), 91	<code>on_peripheral_security_request</code> ( <code>blatann.gap.smp.SecurityManager</code> property), 73
<code>on_discovery_complete()</code> (in module <code>blatann.examples.peripheral</code> ), 53	<code>on_peripheral_security_request()</code> (in module <code>blatann.examples.central</code> ), 50
<code>on_driver_event()</code> ( <code>blatann.nrf.nrf_driver.NrfDriverObserver</code> method), 116	<code>on_phy_updated</code> ( <code>blatann.peer.Peer</code> property), 161
<code>on_gatts_subscription_state_changed()</code> (in module <code>blatann.examples.peripheral</code> ), 53	<code>on_read</code> ( <code>blatann.gatt.gatts.GattsCharacteristic</code> property), 86
<code>on_hex_conversion_characteristic_write()</code> (in module <code>blatann.examples.peripheral</code> ), 53	<code>on_read</code> ( <code>blatann.gatt.gatts_attribute.GattsAttribute</code> property), 89
<code>on_local_time_info_updated</code> ( <code>blatann.services.current_time.service.CurrentTimeClient</code> property), 127	<code>on_read()</code> (in module <code>blatann.examples.peripheral_descriptors</code> ), 56
<code>on_local_time_info_write</code> ( <code>blatann.services.current_time.service.CurrentTimeServer</code> property), 126	<code>on_read_complete</code> ( <code>blatann.gatt.gattc.GattcCharacteristic</code> property), 79
<code>on_local_time_info_write()</code> (in module <code>blatann.examples.peripheral_current_time_service</code> ), 55	<code>on_read_complete</code> ( <code>blatann.gatt.gattc_attribute.GattcAttribute</code> property), 83
<code>on_mtu_exchange_complete</code> ( <code>blatann.peer.Peer</code> property), 161	<code>on_read_complete</code> ( <code>blatann.gatt.reader.GattcReader</code> property), 90
<code>on_mtu_size_update()</code> (in module <code>blatann.examples.central_uart_service</code> ), 49	<code>on_reference_info_updated</code> ( <code>blatann.services.current_time.service.CurrentTimeClient</code> property), 127
<code>on_mtu_size_update()</code> (in module <code>blatann.examples.peripheral_uart_service</code> ), 58	<code>on_rssi_changed</code> ( <code>blatann.peer.Peer</code> property), 161
<code>on_mtu_size_updated</code> ( <code>blatann.peer.Peer</code> property), 161	<code>on_rssi_changed()</code> (in module <code>blatann.examples.peripheral_rssi</code> ), 58
<code>on_notification_received</code> ( <code>blatann.gatt.gattc.GattcCharacteristic</code> property), 79	<code>on_scan_received</code> ( <code>blatann.gap.scanning.Scanner</code> property), 71
<code>on_notify_complete</code> ( <code>blatann.gatt.gatts.GattsCharacteristic</code> property), 87	<code>on_scan_timeout</code> ( <code>blatann.gap.scanning.Scanner</code> property), 71
<code>on_pairing_complete</code> ( <code>blatann.gap.smp.SecurityManager</code> property), 73	<code>on_security_level_changed</code> ( <code>blatann.gap.smp.SecurityManager</code> property), 73
<code>on_pairing_request_rejected</code> ( <code>blatann.gap.smp.SecurityManager</code> property), 74	<code>on_security_level_changed()</code> (in module <code>blatann.examples.peripheral</code> ), 53
<code>on_passkey_display()</code> (in module <code>blatann.examples.peripheral</code> ), 54	<code>on_security_level_changed()</code> (in module <code>blatann.examples.peripheral_glucose_service</code> ), 57
<code>on_passkey_display_required</code> ( <code>blatann.gap.smp.SecurityManager</code> property),	<code>on_subscription_change</code> ( <code>blatann.gatt.gatts.GattsCharacteristic</code> property), 87
	<code>on_time_char_read()</code> (in module <code>blatann.examples.peripheral</code> ), 53
	<code>on_tx_complete()</code> (in module <code>blatann.examples.peripheral_uart_service</code> ), 59

on\_write (*blatann.gatt.gatts.GattsCharacteristic property*), 86  
 on\_write (*blatann.gatt.gatts\_attribute.GattsAttribute property*), 89  
 on\_write\_complete (*blatann.gatt.gattc.GattcCharacteristic property*), 79  
 on\_write\_complete (*blatann.gatt.gattc\_attribute.GattcAttribute property*), 83  
 on\_write\_complete (*blatann.gatt.writer.GattcWriter property*), 91  
 on\_write\_complete (*blatann.services.nordic\_uart.service.NordicUartClient property*), 140  
 on\_write\_complete (*blatann.services.nordic\_uart.service.NordicUartServer property*), 140  
 one\_mbps (*blatann.gap.gap\_types.Phy attribute*), 69  
 one\_mbps (*blatann.nrf.nrf\_types.enums.BLEGapPhy attribute*), 104  
 OOB (*blatann.nrf.nrf\_types.enums.BLEGapAuthKeyType attribute*), 105  
 oob\_not\_available (*blatann.nrf.nrf\_types.enums.BLEGapSecStatus attribute*), 105  
 OPEN (*blatann.gap.smp.SecurityLevel attribute*), 72  
 OPEN (*blatann.nrf.nrf\_types.smp.BLEGapSecModeType attribute*), 114  
 open() (*blatann.device.BleDevice method*), 152  
 open() (*blatann.nrf.nrf\_driver.NrfDriver method*), 117  
 operand\_not\_supported (*blatann.services.glucose.racp.RacpResponseCode attribute*), 138  
 operator\_not\_supported (*blatann.services.glucose.racp.RacpResponseCode attribute*), 138  
 option\_flag (*blatann.nrf.nrf\_types.config.BleOptConnEventExtension attribute*), 98  
 option\_flag (*blatann.nrf.nrf\_types.config.BleOptGapAuthPayloadTimeout attribute*), 99  
 option\_flag (*blatann.nrf.nrf\_types.config.BleOptGapChannelMap attribute*), 99  
 option\_flag (*blatann.nrf.nrf\_types.config.BleOptGapCompatModel attribute*), 99  
 option\_flag (*blatann.nrf.nrf\_types.config.BleOptGapLocalConnLatency attribute*), 99  
 option\_flag (*blatann.nrf.nrf\_types.config.BleOptGapPasskey attribute*), 99  
 option\_flag (*blatann.nrf.nrf\_types.config.BleOptGapScanRequestReport attribute*), 99  
 option\_flag (*blatann.nrf.nrf\_types.config.BleOptGapSlaveLatencyDisable attribute*), 100  
 option\_flag (*blatann.nrf.nrf\_types.config.BleOption attribute*), 98  
 option\_flag (*blatann.nrf.nrf\_types.config.BleOptPaLna attribute*), 99  
 ots\_feature (*blatann.bt\_sig.uuids.CharacteristicUuid attribute*), 42  
 out\_of\_range (*blatann.services.current\_time.data\_types.TimeAccuracy attribute*), 124  
 outdoor\_sports\_act (*blatann.bt\_sig.assigned\_numbers.Appearance attribute*), 30  
 outdoor\_sports\_act\_loc\_and\_nav\_disp (*blatann.bt\_sig.assigned\_numbers.Appearance attribute*), 30  
 outdoor\_sports\_act\_loc\_and\_nav\_pod (*blatann.bt\_sig.assigned\_numbers.Appearance attribute*), 30  
 outdoor\_sports\_act\_loc\_disp (*blatann.bt\_sig.assigned\_numbers.Appearance attribute*), 30  
 outdoor\_sports\_act\_loc\_pod (*blatann.bt\_sig.assigned\_numbers.Appearance attribute*), 30  
 outside (*blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor attribute*), 25  
 ozone\_concentration (*blatann.bt\_sig.uuids.CharacteristicUuid attribute*), 42

## P

pa\_lna (*blatann.nrf.nrf\_types.config.BleOptionFlag attribute*), 98  
 pair() (*blatann.gap.smp.SecurityManager method*), 74  
 PAIRING (*blatann.event\_args.SecurityProcess attribute*), 155  
 pairing\_in\_process (*blatann.gap.smp.SecurityManager property*), 74  
 pairing\_not\_supp (*blatann.nrf.nrf\_types.enums.BLEGapSecStatus attribute*), 105  
 PairingCompleteEventArgs (*class in blatann.event\_args*), 155  
 PairingPolicy (*class in blatann.gap.smp*), 72  
 PairingRejectedEventArgs (*class in blatann.event\_args*), 156  
 PairingRejectedReason (*class in blatann.event\_args*), 156  
 pairing\_with\_unit\_key\_unsupported (*blatann.nrf.nrf\_types.enums.BLEHci attribute*), 101  
 parameter\_out\_of\_mandatory\_range (*blatann.nrf.nrf\_types.enums.BLEHci attribute*), 101

parent ( <i>blatann.gatt.gatts_attribute.GattsAttribute</i> property), 88		<i>tann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42	
parent_group_object_id ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42		percentage ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 27	
particulate_matter_10_concentration ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42		percentage_8 ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42	
particulate_matter_1_concentration ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42		period_beats_per_minute ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 27	
particulate_matter_2_5_concentration ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42		periph ( <i>blatann.nrf.nrf_types.enums.BLEGapRoles</i> attribute), 104	
PASSKEY ( <i>blatann.nrf.nrf_types.enums.BLEGapAuthKeyType</i> attribute), 105		Peripheral (class in <i>blatann.peer</i> ), 164	
passkey_entry_failed ( <i>blatann.nrf.nrf_types.enums.BLEGapSecStatus</i> attribute), 105		peripheral_preferred_connection_parameters ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42	
PasskeyDisplayEventArgs (class in <i>blatann.event_args</i> ), 155		peripheral_privacy_flag ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 43	
PasskeyEntryEventArgs (class in <i>blatann.event_args</i> ), 155		PeripheralConnectionWaitable (class in <i>blatann.waitables.connection_waitable</i> ), 147	
path ( <i>blatann.nrf.nrf_types.config.BleOptConnEventExtention</i> attribute), 98		PeripheralSecurityRequestEventArgs (class in <i>blatann.event_args</i> ), 156	
path ( <i>blatann.nrf.nrf_types.config.BleOptGapAuthPayloadTimeout</i> attribute), 100		PeripheralSecurityRequestEventArgs.Response (class in <i>blatann.event_args</i> ), 156	
path ( <i>blatann.nrf.nrf_types.config.BleOptGapChannelMap</i> attribute), 99		permeability_henry_per_metre ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 27	
path ( <i>blatann.nrf.nrf_types.config.BleOptGapCompatModel</i> attribute), 99		permittivity_farad_per_metre ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 27	
path ( <i>blatann.nrf.nrf_types.config.BleOptGapLocalConnLatency</i> attribute), 99		phone ( <i>blatann.bt_sig.assigned_numbers.Appearance</i> attribute), 29	
path ( <i>blatann.nrf.nrf_types.config.BleOptGapPasskey</i> attribute), 99		phone_alert_status ( <i>blatann.bt_sig.uuids.ServiceUuid</i> attribute), 33	
path ( <i>blatann.nrf.nrf_types.config.BleOptGapScanRequestReply</i> attribute), 99		Report (class in <i>blatann.gap.gap_types</i> ), 69	
path ( <i>blatann.nrf.nrf_types.config.BleOptGapSlaveLatency</i> attribute), 100		phy_channel ( <i>blatann.peer.Peer</i> property), 161	
path ( <i>blatann.nrf.nrf_types.config.BleOption</i> attribute), 98		PhysicalActivityMonitor ( <i>blatann.bt_sig.uuids.ServiceUuid</i> attribute), 33	
path ( <i>blatann.nrf.nrf_types.config.BleOptPaLna</i> attribute), 99		physical_activity_monitor_control_point ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 43	
pdu_invalid ( <i>blatann.nrf.nrf_types.enums.BLEGapSecStatus</i> attribute), 105		physical_activity_monitor_features ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 43	
Peer (class in <i>blatann.peer</i> ), 159		physical_activity_session_descriptor ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 43	
peer_address_matches_or_resolves() ( <i>blatann.gap.bond_db.BondDbEntry</i> method), 66		PhyUpdatedEventArgs (class in <i>blatann.event_args</i> ), 155	
PeerAddress (class in <i>blatann.gap.gap_types</i> ), 69		PickleDatabaseStrategy (class in <i>blatann.gap.default_bond_db</i> ), 68	
PeerState (class in <i>blatann.peer</i> ), 159		pin_or_key_missing ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 43	
per_mille ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 27			
perceived_lightness ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 42			

<code>tann.nrf.nrf_types.enums.BLEHci</code> attribute), 101	<code>tribute</code> ), 43
<code>plane_angle_degree</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 27	<code>power_watt</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 28
<code>plane_angle_minute</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 27	<code>preferred_connection_params</code> ( <code>blatann.peer.Peer</code> property), 160
<code>plane_angle_radian</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 28	<code>preferred_mtu_size</code> ( <code>blatann.peer.Peer</code> property), 160
<code>plane_angle_second</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 28	<code>preferred_peripheral_connection_params</code> ( <code>blatann.gap.generic_access_service.GenericAccessService</code> property), 71
<code>playback_speed</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>preferred_phy</code> ( <code>blatann.peer.Peer</code> property), 161
<code>playing_order</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>preferred_units</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43
<code>playing_orders_supported</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>premixed_insulin</code> ( <code>blatann.services.glucose.data_types.MedicationType</code> attribute), 132
<code>plx_continuous_measurement</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>prep_write_req</code> ( <code>blatann.nrf.nrf_types.enums.BLEGattsWriteOperation</code> attribute), 107
<code>plx_features</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>prepare_queue_full</code> ( <code>blatann.nrf.nrf_types.enums.BLEGattStatusCode</code> attribute), 106
<code>plx_spot_check_measurement</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>prepare_write_req</code> ( <code>blatann.nrf.nrf_types.enums.BLEGattWriteOperation</code> attribute), 106
<code>pnp_id</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>prepared_cancel</code> ( <code>blatann.nrf.nrf_types.enums.BLEGattExecWriteFlag</code> attribute), 107
<code>PnpId</code> (class in <code>blatann.services.device_info.data_types</code> ), 128	<code>prepared_write</code> ( <code>blatann.nrf.nrf_types.enums.BLEGattExecWriteFlag</code> attribute), 107
<code>PnpVendorSource</code> (class in <code>blatann.services.device_info.data_types</code> ), 128	<code>preprandial</code> ( <code>blatann.services.glucose.data_types.MealType</code> attribute), 131
<code>pollen_concentration</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>presentation_format</code> ( <code>blatann.bt_sig.uuids.DescriptorUuid</code> attribute), 31
<code>POS_INFINITY</code> ( <code>blatann.services.ble_data_types.SFloat</code> attribute), 144	<code>presentation_format</code> ( <code>blatann.gatt.gatts.GattsCharacteristic</code> property), 86
<code>position_2d</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>PresentationFormat</code> (class in <code>blatann.gatt</code> ), 78
<code>position_3d</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>pressure</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43
<code>position_quality</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>pressure_bar</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 28
<code>postprandial</code> ( <code>blatann.services.glucose.data_types.MealType</code> attribute), 131	<code>pressure_millimetre_of_mercury</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 28
<code>power</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>pressure_pascal</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 28
<code>power_specification</code> ( <code>blatann.bt_sig.uuids.CharacteristicUuid</code> attribute), 43	<code>pressure_pound_force_per_square_inch</code> ( <code>blatann.bt_sig.assigned_numbers.Units</code> attribute), 28
	<code>PRIMARY</code> ( <code>blatann.gatt.ServiceType</code> attribute), 77



- `primary_service` (*blatann.bt\_sig.uuids.DeclarationUuid* attribute), 31
- `private_address_resolves()` (in module *blatann.gap.smp\_crypto*), 76
- `procedure_not_completed` (*blatann.services.glucose.racp.RacpResponseCode* attribute), 138
- `protocol_mode` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- `public` (*blatann.nrf.nrf\_types.gap.BLEGapAddrTypes* attribute), 108
- `public_target_address` (*blatann.gap.advertise\_data.AdvertisingData.Types* attribute), 60
- `public_target_address` (*blatann.nrf.nrf\_types.gap.BLEAdvData.Types* attribute), 109
- `published_audio_capabilities` (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 33
- `pulse_oximeter` (*blatann.bt\_sig.assigned\_numbers.Appearance* attribute), 30
- `pulse_oximeter` (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 33
- `pulse_oximeter_fingertip` (*blatann.bt\_sig.assigned\_numbers.Appearance* attribute), 30
- `pulse_oximeter_wrist_worn` (*blatann.bt\_sig.assigned\_numbers.Appearance* attribute), 30
- `pulse_oximetry_control_point` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- Q**
- `QUEUE_CLEARED` (*blatann.event\_args.GattOperationCompleteReason* attribute), 154
- `QueuedTasksManagerBase` (class in *blatann.utils.queued\_tasks\_manager*), 147
- `QueuedTasksManagerBase.TaskFailure` (class in *blatann.utils.queued\_tasks\_manager*), 147
- R**
- `RacpCommand` (class in *blatann.services.glucose.racp*), 138
- `RacpOpcode` (class in *blatann.services.glucose.racp*), 137
- `RacpOperator` (class in *blatann.services.glucose.racp*), 137
- `RacpResponse` (class in *blatann.services.glucose.racp*), 139
- `RacpResponseCode` (class in *blatann.services.glucose.racp*), 138
- `radiance_watt_per_square_metre_steradian` (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 28
- `radiant_intensity_watt_per_steradian` (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 28
- `radio_time_signal` (*blatann.services.current\_time.data\_types.TimeSource* attribute), 123
- `rainfall` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- `RAND_INVALID` (*blatann.nrf.nrf\_types.smp.BLEGapMasterId* attribute), 115
- `RAND_LEN` (*blatann.nrf.nrf\_types.smp.BLEGapMasterId* attribute), 115
- `random_private_non_resolvable` (*blatann.nrf.nrf\_types.gap.BLEGapAddrTypes* attribute), 108
- `random_private_resolvable` (*blatann.nrf.nrf\_types.gap.BLEGapAddrTypes* attribute), 108
- `random_static` (*blatann.nrf.nrf\_types.gap.BLEGapAddrTypes* attribute), 108
- `random_target_address` (*blatann.gap.advertise\_data.AdvertisingData.Types* attribute), 60
- `random_target_address` (*blatann.nrf.nrf\_types.gap.BLEAdvData.Types* attribute), 109
- `rapid_acting_insulin` (*blatann.services.glucose.data\_types.MedicationType* attribute), 132
- `rc_feature` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- `rc_settings` (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- `read()` (*blatann.gatt.gattc.GattcCharacteristic* method), 80
- `read()` (*blatann.gatt.gattc\_attribute.GattcAttribute* method), 83
- `read()` (*blatann.gatt.managers.GattcOperationManager* method), 90
- `read()` (*blatann.gatt.reader.GattcReader* method), 90
- `read()` (*blatann.services.battery.service.BatteryClient* method), 121
- `read_in_process` (*blatann.gatt.gatts\_attribute.GattsAttribute* property), 89
- `read_not_permitted` (*blatann.nrf.nrf\_types.enums.BLEGattStatusCode* attribute), 106
- `read_time()` (*blatann.services.current\_time.service.CurrentTimeClient*

- method*), 127
- readable (*blatann.gattc.GattcCharacteristic* property), 78
- ReadCompleteEventArgs (*class in blatann.event\_args*), 157
- Reason (*blatann.event\_args.NotificationCompleteEventArgs* attribute), 157
- reconnection\_address (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- reconnection\_configuration (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 33
- reconnection\_configuration\_control\_point (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- record\_access\_control\_point (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- record\_count() (*blatann.services.glucose.database.BasicGlucoseDatabase* method), 137
- record\_count() (*blatann.services.glucose.database.IGlucoseDatabase* method), 136
- reference\_time\_information (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- reference\_time\_update (*blatann.bt\_sig.uuids.ServiceUuid* attribute), 33
- ReferenceTimeInfo (*class in blatann.services.current\_time.data\_types*), 125
- refractive\_index (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 28
- register() (*blatann.event\_type.Event* method), 158
- registered\_user\_characteristic (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- reject (*blatann.event\_args.PeripheralSecurityRequestEventArgs.Response* attribute), 156
- reject() (*blatann.event\_args.PeripheralSecurityRequestEventArgs* method), 156
- reject\_all\_requests (*blatann.gap.smp.PairingPolicy* attribute), 72
- reject\_bonded\_device\_repairing\_requests (*blatann.gap.smp.PairingPolicy* attribute), 72
- reject\_bonded\_peripheral\_requests (*blatann.gap.smp.PairingPolicy* attribute), 72
- reject\_conn\_param\_requests() (*blatann.peer.Peripheral* method), 164
- reject\_new\_pairing\_requests (*blatann.gap.smp.PairingPolicy* attribute), 72
- reject\_nonbonded\_peripheral\_requests (*blatann.gap.smp.PairingPolicy* attribute), 72
- reject\_peripheral\_requests (*blatann.gap.smp.PairingPolicy* attribute), 72
- relative\_permeability (*blatann.bt\_sig.assigned\_numbers.Units* attribute), 28
- relative\_runtime\_current\_range (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- relative\_runtime\_generic\_level\_range (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 43
- relative\_value\_illuminance\_range (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 44
- relative\_value\_period\_of\_day (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 44
- relative\_value\_temperature\_range (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 44
- relative\_value\_voltage\_range (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 44
- reload() (*blatann.nrf.nrf\_types.smp.BLEGapSecKeyset* method), 116
- remote\_control (*blatann.bt\_sig.assigned\_numbers.Appearance* attribute), 29
- remote\_dev\_termination\_due\_to\_low\_resources (*blatann.nrf.nrf\_types.enums.BLEHci* attribute), 101
- remote\_dev\_termination\_due\_to\_power\_off (*blatann.nrf.nrf\_types.enums.BLEHci* attribute), 101
- remote\_user\_terminated\_connection (*blatann.nrf.nrf\_types.enums.BLEHci* attribute), 101
- removable (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 44
- repeated\_attempts (*blatann.nrf.nrf\_types.enums.BLEGapSecStatus* attribute), 105
- report (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 44
- report\_map (*blatann.bt\_sig.uuids.CharacteristicUuid* attribute), 44
- report\_number\_of\_records (*blatann.services.glucose.racp.RacpOpcode* attribute), 137
- report\_reference (*blatann.bt\_sig.uuids.DescriptorUuid* attribute), 31

report_stored_records	(blatann.services.glucose.racp.RacpOpcode attribute), 137	ringer_control_point	(blatann.bt_sig.uuids.CharacteristicUuid attribute), 44
repr_format() (in module blatann.utils), 146		ringer_setting	(blatann.bt_sig.uuids.CharacteristicUuid attribute), 44
request_not_supported	(blatann.nrf.nrf_types.enums.BLEGattStatusCode attribute), 106	reserved_data_value	(blatann.bt_sig.uuids.CharacteristicUuid attribute), 44
RESERVED (blatann.services.ble_data_types.SFloat.Reserved attribute), 144		rpc_decode	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
resolvable_private_address_only	(blatann.bt_sig.uuids.CharacteristicUuid attribute), 44	rpc_encode	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
resolve() (blatann.event_args.PasskeyEntryEventArgs method), 155		rpc_h5_transport	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
resolved_address	(blatann.gap.advertise_data.ScanReport property), 62	rpc_h5_transport_already_closed	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
resolved_peer_address()	(blatann.gap.bond_db.BondDbEntry method), 66	rpc_h5_transport_already_open	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
resources	(blatann.nrf.nrf_types.enums.NrfError attribute), 102	rpc_h5_transport_header_checksum	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
response_code (blatann.services.glucose.racp.RacpOpcode attribute), 137		rpc_h5_transport_internal_error	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
resting_heart_rate	(blatann.bt_sig.uuids.CharacteristicUuid attribute), 44	rpc_h5_transport_no_response	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
result_above_range	(blatann.services.glucose.data_types.SensorStatusType attribute), 132	rpc_h5_transport_packet_checksum	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
result_below_range	(blatann.services.glucose.data_types.SensorStatusType attribute), 132	rpc_h5_transport_slip_calculated_payload_size	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
rfu (blatann.bt_sig.assigned_numbers.Format attribute), 23		rpc_h5_transport_slip_decoding	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
rfu_range1_begin	(blatann.nrf.nrf_types.enums.BLEGattStatusCode attribute), 107	rpc_h5_transport_slip_payload_size	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
rfu_range1_end	(blatann.nrf.nrf_types.enums.BLEGattStatusCode attribute), 107	rpc_h5_transport_state	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
rfu_range2_begin	(blatann.nrf.nrf_types.enums.BLEGattStatusCode attribute), 107	rpc_invalid_argument	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
rfu_range2_end	(blatann.nrf.nrf_types.enums.BLEGattStatusCode attribute), 107	rpc_invalid_state	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
rfu_range3_begin	(blatann.nrf.nrf_types.enums.BLEGattStatusCode attribute), 107	rpc_no_response	(blatann.nrf.nrf_types.enums.NrfError attribute), 103
rfu_range3_end	(blatann.nrf.nrf_types.enums.BLEGattStatusCode attribute), 107		
right (blatann.bt_sig.assigned_numbers.NamespaceDescriptor attribute), 23			



- 103
- `rpc_send` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 103
- `rpc_serial_port` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 103
- `rpc_serial_port_already_closed` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 103
- `rpc_serial_port_already_open` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 103
- `rpc_serial_port_internal_error` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 103
- `rpc_serial_port_state` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 103
- `rpc_serialization_transport` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 103
- `rpc_serialization_transport_already_closed` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 103
- `rpc_serialization_transport_already_open` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 103
- `rpc_serialization_transport_invalid_state` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 103
- `rpc_serialization_transport_no_response` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 103
- `rsc_feature` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `rsc_measurement` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `rss` (`blatann.peer.Peer` property), 160
- `run()` (`blatann.examples.peripheral.CountingCharacteristicThread` property), 149
- `running_speed_and_cadence` (`blatann.bt_sig.uuids.ServiceUuid` attribute), 33
- `running_walking_sensor` (`blatann.bt_sig.assigned_numbers.Appearance` attribute), 30
- `running_walking_sensor_in_shoe` (`blatann.bt_sig.assigned_numbers.Appearance` attribute), 30
- `running_walking_sensor_on_hip` (`blatann.bt_sig.assigned_numbers.Appearance` attribute), 30
- `running_walking_sensor_on_shoe` (`blatann.bt_sig.assigned_numbers.Appearance` attribute), 30
- `sample_size_insufficient` (`blatann.services.glucose.data_types.SensorStatusType` attribute), 132
- `SampleLocation` (class in `blatann.services.glucose.data_types`), 131
- `saturday` (`blatann.services.ble_data_types.DayOfWeek` attribute), 145
- `save()` (`blatann.gap.bond_db.BondDatabaseLoader` method), 67
- `save()` (`blatann.gap.default_bond_db.DatabaseStrategy` method), 67
- `save()` (`blatann.gap.default_bond_db.DefaultBondDatabaseLoader` method), 68
- `save()` (`blatann.gap.default_bond_db.JsonDatabaseStrategy` method), 67
- `save()` (`blatann.gap.default_bond_db.PickleDatabaseStrategy` method), 68
- `sc_control_point` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `scan` (`blatann.nrf.nrf_types.enums.BLEGapTimeoutSrc` attribute), 104
- `scan_and_connect()` (`blatann.examples.central_event_driven.ConnectionManager` method), 52
- `scan_interval_window` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `scan_parameters` (`blatann.bt_sig.uuids.ServiceUuid` attribute), 33
- `scan_params_setup()` (`blatann.nrf.nrf_driver.NrfDriver` method), 117
- `scan_refresh` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `scan_reports` (`blatann.waitables.scan_waitable.ScanFinishedWaitable` property), 149
- `scan_response` (`blatann.nrf.nrf_types.enums.BLEGapAdvType` attribute), 104
- `scannable_undirected` (`blatann.nrf.nrf_types.enums.BLEGapAdvType` attribute), 104
- `ScanFinishedWaitable` (class in `blatann.waitables.scan_waitable`), 149
- `Scanner` (class in `blatann.gap.scanning`), 71
- `ScanParameters` (class in `blatann.gap.scanning`), 71
- `ScanReport` (class in `blatann.gap.advertise_data`), 62
- `ScanReportCollection` (class in `blatann.gap.advertise_data`), 63
- `sccd` (`blatann.bt_sig.uuids.DescriptorUuid` attribute), 31

- `sccd` (`blatann.gatt.gatts.GattsCharacteristic` property), 86
- `scientific_temperature_celsius` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `search_control_point` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `search_results_object_id` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `SECONDARY` (`blatann.gatt.ServiceType` attribute), 77
- `secondary_service` (`blatann.bt_sig.uuids.DeclarationUuid` attribute), 31
- `secondary_time_zone` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `security_level` (`blatann.gap.smp.SecurityManager` property), 74
- `security_manager_oob_flags` (`blatann.gap.advertise_data.AdvertisingData.Types` attribute), 60
- `security_manager_oob_flags` (`blatann.nrf.nrf_types.gap.BLEAdvData.Types` attribute), 109
- `security_manager_tk_value` (`blatann.gap.advertise_data.AdvertisingData.Types` attribute), 60
- `security_manager_tk_value` (`blatann.nrf.nrf_types.gap.BLEAdvData.Types` attribute), 109
- `security_params` (`blatann.gap.smp.SecurityManager` property), 74
- `security_params_setup()` (`blatann.nrf.nrf_driver.NrfDriver` method), 117
- `SecurityLevel` (class in `blatann.gap.smp`), 72
- `SecurityLevelChangedEventArgs` (class in `blatann.event_args`), 155
- `SecurityManager` (class in `blatann.gap.smp`), 73
- `SecurityParameters` (class in `blatann.gap.smp`), 72
- `SecurityProcess` (class in `blatann.event_args`), 155
- `sedentary_interval_notification` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `seeking_speed` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `self` (`blatann.services.glucose.data_types.TesterType` attribute), 132
- `sensor_location` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `sensor_malfunction` (`blatann.services.glucose.data_types.SensorStatusType` attribute), 132
- `sensor_malfunction_detection` (`blatann.services.glucose.data_types.GlucoseFeatureType` attribute), 133
- `sensor_read_interrupt_detection` (`blatann.services.glucose.data_types.GlucoseFeatureType` attribute), 133
- `sensor_read_interrupted` (`blatann.services.glucose.data_types.SensorStatusType` attribute), 133
- `sensor_result_high_low_detection` (`blatann.services.glucose.data_types.GlucoseFeatureType` attribute), 133
- `sensor_sample_size` (`blatann.services.glucose.data_types.GlucoseFeatureType` attribute), 133
- `sensor_temp_high` (`blatann.services.glucose.data_types.SensorStatusType` attribute), 133
- `sensor_temp_high_low_detection` (`blatann.services.glucose.data_types.GlucoseFeatureType` attribute), 133
- `sensor_temp_low` (`blatann.services.glucose.data_types.SensorStatusType` attribute), 133
- `SensorStatus` (class in `blatann.services.glucose.data_types`), 133
- `SensorStatusType` (class in `blatann.services.glucose.data_types`), 132
- `sequence_number` (`blatann.services.glucose.racp.FilterType` attribute), 138
- `serial_number_string` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `serial_port` (`blatann.nrf.nrf_driver.NrfDriver` property), 117
- `serial_port_desc_array_to_list()` (in module `blatann.nrf.nrf_driver_types`), 120
- `SERVER_DISCONNECTED` (`blatann.event_args.GattOperationCompleteReason` attribute), 154
- `server_supported_features` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 44
- `Service` (class in `blatann.gatt`), 77
- `service_128bit_uuid_complete` (`blatann.gap.advertise_data.AdvertisingData.Types` attribute), 60
- `service_128bit_uuid_complete` (`blatann.nrf.nrf_types.gap.BLEAdvData.Types` attribute), 109
- `service_128bit_uuid_more_available` (`blatann.gap.advertise_data.AdvertisingData.Types` attribute), 60

<i>attribute</i> ), 60	<i>service_required</i> (blatann.bt_sig.uuids.CharacteristicUuid <i>attribute</i> ), 44
<i>service_128bit_uuid_more_available</i> (blatann.nrf.nrf_types.gap.BLEAdvData.Types <i>attribute</i> ), 109	<i>service_secondary</i> (blatann.nrf.nrf_types.generic.BLEUUID.Standard <i>attribute</i> ), 113
<i>service_16bit_uuid_complete</i> (blatann.gap.advertise_data.AdvertisingData.Types <i>attribute</i> ), 60	<i>service_uuids</i> (blatann.gap.advertise_data.AdvertisingData <i>property</i> ), 61
<i>service_16bit_uuid_complete</i> (blatann.nrf.nrf_types.gap.BLEAdvData.Types <i>attribute</i> ), 109	<i>services</i> (blatann.gatt.gattc.GattcDatabase <i>property</i> ), 82
<i>service_16bit_uuid_more_available</i> (blatann.gap.advertise_data.AdvertisingData.Types <i>attribute</i> ), 60	<i>services</i> (blatann.gatt.gatts.GattsDatabase <i>property</i> ), 88
<i>service_16bit_uuid_more_available</i> (blatann.nrf.nrf_types.gap.BLEAdvData.Types <i>attribute</i> ), 109	<i>ServiceType</i> (class in blatann.gatt), 77
<i>service_32bit_uuid_complete</i> (blatann.gap.advertise_data.AdvertisingData.Types <i>attribute</i> ), 60	<i>ServiceUuid</i> (class in blatann.bt_sig.uuids), 32
<i>service_32bit_uuid_complete</i> (blatann.nrf.nrf_types.gap.BLEAdvData.Types <i>attribute</i> ), 109	<i>set()</i> (blatann.services.device_info.service.DisServer <i>method</i> ), 129
<i>service_32bit_uuid_more_available</i> (blatann.gap.advertise_data.AdvertisingData.Types <i>attribute</i> ), 60	<i>set_advertise_data()</i> (blatann.gap.advertising.Advertiser <i>method</i> ), 65
<i>service_32bit_uuid_more_available</i> (blatann.nrf.nrf_types.gap.BLEAdvData.Types <i>attribute</i> ), 109	<i>set_battery_level()</i> (blatann.services.battery.service.BatteryServer <i>method</i> ), 121
<i>service_array_to_list()</i> (in module blatann.nrf.nrf_driver_types), 119	<i>set_channel_mask()</i> (blatann.gap.advertising.Advertiser <i>method</i> ), 64
<i>service_changed</i> (blatann.bt_sig.uuids.CharacteristicUuid <i>attribute</i> ), 44	<i>set_conn_param_request_handler()</i> (blatann.peer.Peripheral <i>method</i> ), 164
<i>service_data</i> (blatann.gap.advertise_data.AdvertisingData <i>property</i> ), 61	<i>set_connection_parameters()</i> (blatann.peer.Peer <i>method</i> ), 162
<i>service_data</i> (blatann.gap.advertise_data.AdvertisingData.Types <i>attribute</i> ), 60	<i>set_default_advertise_params()</i> (blatann.gap.advertising.Advertiser <i>method</i> ), 65
<i>service_data</i> (blatann.nrf.nrf_types.gap.BLEAdvData.Types <i>attribute</i> ), 109	<i>set_default_peripheral_connection_params()</i> (blatann.device.BleDevice <i>method</i> ), 153
<i>service_data_128bit_uuid</i> (blatann.gap.advertise_data.AdvertisingData.Types <i>attribute</i> ), 61	<i>set_default_scan_params()</i> (blatann.gap.scanning.Scanner <i>method</i> ), 71
<i>service_data_128bit_uuid</i> (blatann.nrf.nrf_types.gap.BLEAdvData.Types <i>attribute</i> ), 110	<i>set_default_security_params()</i> (blatann.device.BleDevice <i>method</i> ), 153
<i>service_data_32bit_uuid</i> (blatann.gap.advertise_data.AdvertisingData.Types <i>attribute</i> ), 61	<i>set_features()</i> (blatann.services.glucose.service.GlucoseServer <i>method</i> ), 139
<i>service_data_32bit_uuid</i> (blatann.nrf.nrf_types.gap.BLEAdvData.Types <i>attribute</i> ), 110	<i>set_firmware_revision()</i> (blatann.services.device_info.service.DisServer <i>method</i> ), 129
<i>service_primary</i> (blatann.nrf.nrf_types.generic.BLEUUID.Standard <i>attribute</i> ), 113	<i>set_hardware_revision()</i> (blatann.services.device_info.service.DisServer <i>method</i> ), 129
	<i>set_identity_resolving_key</i> (blatann.bt_sig.uuids.CharacteristicUuid <i>attribute</i> ), 44
	<i>set_local_time_info()</i> (blatann.services.current_time.service.CurrentTimeServer <i>method</i> ), 126
	<i>set_manufacturer_name()</i> (bla-

<code>tann.services.device_info.service.DisServer</code>	<code>attribute</code> ), 60
<code>method</code> ), 129	<code>short_local_name</code> (bla-
<code>set_member_lock</code> (bla-	<code>tann.nrf.nrf_types.gap.BLEAdvData.Types</code>
<code>tann.bt_sig.uuids.CharacteristicUuid</code>	<code>attribute</code> ), 109
<code>tribute</code> ), 44	<code>SIGN_OR_ENCRYPT</code> (bla-
<code>set_member_rank</code> (bla-	<code>tann.nrf.nrf_types.smp.BLEGapSecModeType</code>
<code>tann.bt_sig.uuids.CharacteristicUuid</code>	<code>attribute</code> ), 114
<code>tribute</code> ), 44	<code>SIGN_OR_ENCRYPT_MITM</code> (bla-
<code>set_model_number()</code> (bla-	<code>tann.nrf.nrf_types.smp.BLEGapSecModeType</code>
<code>tann.services.device_info.service.DisServer</code>	<code>attribute</code> ), 114
<code>method</code> ), 129	<code>sign_write_cmd</code> (bla-
<code>set_pnp_id()</code> (blatann.services.device_info.service.DisServer	<code>tann.nrf.nrf_types.enums.BLEGattsWriteOperation</code>
<code>method</code> ), 129	<code>attribute</code> ), 107
<code>set_privacy_settings()</code> (blatann.device.BleDevice	<code>signed</code> (blatann.services.ble_data_types.SignedIntegerBase
<code>method</code> ), 154	<code>attribute</code> ), 142
<code>set_reference_info()</code> (bla-	<code>signed</code> (blatann.services.ble_data_types.UnsignedIntegerBase
<code>tann.services.current_time.service.CurrentTimeServer</code>	<code>attribute</code> ), 142
<code>method</code> ), 126	<code>signed_write_cmd</code> (bla-
<code>set_regulatory_certifications()</code> (bla-	<code>tann.nrf.nrf_types.enums.BLEGattWriteOperation</code>
<code>tann.services.device_info.service.DisServer</code>	<code>attribute</code> ), 106
<code>method</code> ), 129	<code>SignedIntegerBase</code> (class in bla-
<code>set_security_params()</code> (bla-	<code>tann.services.ble_data_types</code> ), 142
<code>tann.gap.smp.SecurityManager</code> <code>method</code> ),	<code>simple_pairing_hash_c</code> (bla-
74	<code>tann.gap.advertise_data.AdvertisingData.Types</code>
<code>set_serial_number()</code> (bla-	<code>attribute</code> ), 60
<code>tann.services.device_info.service.DisServer</code>	<code>simple_pairing_hash_c</code> (bla-
<code>method</code> ), 129	<code>tann.nrf.nrf_types.gap.BLEAdvData.Types</code>
<code>set_software_revision()</code> (bla-	<code>attribute</code> ), 109
<code>tann.services.device_info.service.DisServer</code>	<code>simple_pairing_randimizer_r</code> (bla-
<code>method</code> ), 129	<code>tann.gap.advertise_data.AdvertisingData.Types</code>
<code>set_system_id()</code> (bla-	<code>attribute</code> ), 60
<code>tann.services.device_info.service.DisServer</code>	<code>simple_pairing_randimizer_r</code> (bla-
<code>method</code> ), 129	<code>tann.nrf.nrf_types.gap.BLEAdvData.Types</code>
<code>set_time()</code> (blatann.services.current_time.service.CurrentTimeClient	<code>attribute</code> ), 109
<code>method</code> ), 127	<code>simple_pairng_hash_c256</code> (bla-
<code>set_time()</code> (blatann.services.current_time.service.CurrentTimeServer	<code>tann.gap.advertise_data.AdvertisingData.Types</code>
<code>method</code> ), 126	<code>attribute</code> ), 61
<code>set_tx_power()</code> (blatann.device.BleDevice <code>method</code> ),	<code>simple_pairng_hash_c256</code> (bla-
153	<code>tann.nrf.nrf_types.gap.BLEAdvData.Types</code>
<code>set_value()</code> (blatann.gatt.gatts.GattsCharacteristic	<code>attribute</code> ), 110
<code>method</code> ), 84	<code>simple_pairng_randomizer_r256</code> (bla-
<code>set_value()</code> (blatann.gatt.gatts_attribute.GattsAttribute	<code>tann.gap.advertise_data.AdvertisingData.Types</code>
<code>method</code> ), 89	<code>attribute</code> ), 61
<code>setup_logger()</code> (in module blatann.utils), 146	<code>simple_pairng_randomizer_r256</code> (bla-
<code>sfloat</code> (blatann.bt_sig.assigned_numbers.Format	<code>tann.nrf.nrf_types.gap.BLEAdvData.Types</code>
<code>attribute</code> ), 24	<code>attribute</code> ), 110
<code>SFloat</code> (class in blatann.services.ble_data_types), 143	<code>sink_ase</code> (blatann.bt_sig.uuids.CharacteristicUuid at-
<code>SFloat.ReservedMantissaValues</code> (class in bla-	<code>tribute</code> ), 45
<code>tann.services.ble_data_types</code> ), 144	<code>sink_audio_locations</code> (bla-
<code>short_acting_insulin</code> (bla-	<code>tann.bt_sig.uuids.CharacteristicUuid</code> at-
<code>tann.services.glucose.data_types.MedicationType</code>	<code>tribute</code> ), 45
<code>attribute</code> ), 132	<code>sink_pac</code> (blatann.bt_sig.uuids.CharacteristicUuid at-
<code>short_local_name</code> (bla-	<code>tribute</code> ), 45
<code>tann.gap.advertise_data.AdvertisingData.Types</code>	<code>sint12</code> (blatann.bt_sig.assigned_numbers.Format

- attribute), 23
- sint128 (blatann.bt\_sig.assigned\_numbers.Format attribute), 24
- sint16 (blatann.bt\_sig.assigned\_numbers.Format attribute), 24
- sint24 (blatann.bt\_sig.assigned\_numbers.Format attribute), 24
- sint32 (blatann.bt\_sig.assigned\_numbers.Format attribute), 24
- sint48 (blatann.bt\_sig.assigned\_numbers.Format attribute), 24
- sint64 (blatann.bt\_sig.assigned\_numbers.Format attribute), 24
- sint8 (blatann.bt\_sig.assigned\_numbers.Format attribute), 23
- slave\_connection\_interval\_range (blatann.gap.advertise\_data.AdvertisingData.Types attribute), 60
- slave\_connection\_interval\_range (blatann.nrf.nrf\_types.gap.BLEAdvData.Types attribute), 109
- slave\_latency (blatann.gap.gap\_types.ActiveConnectionParameters attribute), 70
- sleep\_activity\_instantaneous\_data (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- sleep\_activity\_summary\_data (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- smp\_cmd\_unsupported (blatann.nrf.nrf\_types.enums.BLEGapSecStatus attribute), 105
- snack (blatann.services.glucose.data\_types.CarbohydrateType attribute), 131
- snake\_case\_to\_capitalized\_words() (in module blatann.utils), 146
- softdevice\_not\_enabled (blatann.nrf.nrf\_types.enums.NrfError attribute), 102
- software\_revision\_string (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- solicited\_sevice\_uuids\_128bit (blatann.gap.advertise\_data.AdvertisingData.Types attribute), 60
- solicited\_sevice\_uuids\_128bit (blatann.nrf.nrf\_types.gap.BLEAdvData.Types attribute), 109
- solicited\_sevice\_uuids\_16bit (blatann.gap.advertise\_data.AdvertisingData.Types attribute), 60
- solicited\_sevice\_uuids\_16bit (blatann.nrf.nrf\_types.gap.BLEAdvData.Types attribute), 109
- solid\_angle\_steradian (blatann.bt\_sig.assigned\_numbers.Units attribute), 28
- sound\_pressure\_decibel\_spl (blatann.bt\_sig.assigned\_numbers.Units attribute), 28
- source\_ase (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- source\_audio\_locations (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- source\_pac (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- specific\_energy\_joule\_per\_kilogram (blatann.bt\_sig.assigned\_numbers.Units attribute), 28
- specific\_heat\_capacity\_joule\_per\_kilogram\_kelvin (blatann.bt\_sig.assigned\_numbers.Units attribute), 28
- specific\_volume\_cubic\_metre\_per\_kilogram (blatann.bt\_sig.assigned\_numbers.Units attribute), 28
- sport\_type\_for\_aerobic\_and\_anaerobic\_thresholds (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- sports\_watch (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 29
- srvc\_uuid (blatann.nrf.nrf\_types.gatt.BLEGattService attribute), 111
- stair\_climber\_data (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- standard\_time (blatann.services.current\_time.data\_types.DaylightSaving attribute), 123
- start() (blatann.examples.central\_event\_driven.HexConverterTest method), 51
- start() (blatann.gap.advertising.Advertiser method), 65
- start() (blatann.gatt.service\_discovery.DatabaseDiscoverer method), 91
- start() (blatann.utils.Stopwatch method), 146
- start\_rssi\_reporting() (blatann.peer.Peer method), 163
- start\_scan() (blatann.gap.scanning.Scanner method), 71
- start\_time (blatann.utils.Stopwatch property), 146
- status\_flags (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- step\_climber\_data (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- step\_counter\_activity\_summary\_data (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- step\_per\_minute (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45



- `tann.bt_sig.assigned_numbers.Units` attribute), 28
- `stop()` (`blatann.gap.advertising.Advertiser` method), 65
- `stop()` (`blatann.gap.scanning.Scanner` method), 72
- `stop()` (`blatann.utils.Stopwatch` method), 146
- `stop_rssi_reporting()` (`blatann.peer.Peer` method), 164
- `stop_time` (`blatann.utils.Stopwatch` property), 146
- `Stopwatch` (class in `blatann.utils`), 146
- `stride_length` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `string` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `String` (class in `blatann.services.ble_data_types`), 143
- `string_encoding` (`blatann.gatt.Attribute` property), 77
- `string_encoding` (`blatann.gatt.gattc.GattcCharacteristic` property), 79
- `string_encoding` (`blatann.gatt.gatts.GattsCharacteristic` property), 86
- `strip_insertion_error` (`blatann.services.glucose.data_types.SensorStatusType` attribute), 132
- `strip_insertion_error_detection` (`blatann.services.glucose.data_types.GlucoseFeatureType` attribute), 133
- `strip_type_error_detection` (`blatann.services.glucose.data_types.GlucoseFeatureType` attribute), 133
- `stroke_per_minute` (`blatann.bt_sig.assigned_numbers.Units` attribute), 28
- `struct` (`blatann.bt_sig.assigned_numbers.Format` attribute), 24
- `subscribable` (`blatann.gatt.gattc.GattcCharacteristic` property), 79
- `subscribable_indications` (`blatann.gatt.gattc.GattcCharacteristic` property), 79
- `subscribable_notifications` (`blatann.gatt.gattc.GattcCharacteristic` property), 79
- `subscribe()` (`blatann.gatt.gattc.GattcCharacteristic` method), 79
- `subscribed` (`blatann.gatt.gattc.GattcCharacteristic` property), 79
- `SubscriptionState` (class in `blatann.gatt`), 77
- `SubscriptionStateChangeEventArgs` (class in `blatann.event_args`), 157
- `SubscriptionWriteCompleteEventArgs` (class in `blatann.event_args`), 157
- `SUCCESS` (`blatann.event_args.GattOperationCompleteReason` attribute), 154
- `success` (`blatann.nrf.nrf_types.enums.BLEGapSecStatus` attribute), 105
- `success` (`blatann.nrf.nrf_types.enums.BLEGattStatusCode` attribute), 106
- `success` (`blatann.nrf.nrf_types.enums.BLEHci` attribute), 100
- `success` (`blatann.nrf.nrf_types.enums.NrfError` attribute), 101
- `success` (`blatann.services.glucose.racp.RacpResponseCode` attribute), 138
- `sulfur_dioxide_concentration` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `sulfur_hexafluoride_concentration` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `sunday` (`blatann.services.ble_data_types.DayOfWeek` attribute), 145
- `supper` (`blatann.services.glucose.data_types.CarbohydrateType` attribute), 131
- `supplementary` (`blatann.bt_sig.assigned_numbers.NamespaceDescriptor` attribute), 25
- `supported_audio_contexts` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `supported_heart_rate_range` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `supported_inclination_range` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `supported_new_alert_category` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `supported_power_range` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `supported_resistance_level_range` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `supported_speed_range` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `supported_unread_alert_category` (`blatann.bt_sig.uuids.CharacteristicUuid` attribute), 45
- `surface_charge_density_coulomb_per_square_metre` (`blatann.bt_sig.assigned_numbers.Units` attribute), 28
- `surface_density_kilogram_per_square_metre` (`blatann.bt_sig.assigned_numbers.Units` attribute), 28
- `surface_tension_newton_per_metre` (`blatann.bt_sig.assigned_numbers.Units` attribute),

- 28
- `svc_handler_missing` (blatann.nrf.nrf\_types.enums.NrfError attribute), 101
- `SynchronousMonotonicCounter` (class in blatann.utils), 146
- `system_id` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- `SystemId` (class in blatann.services.device\_info.data\_types), 128
- ## T
- `t` (in module blatann.bt\_sig.uuids), 49
- `tag` (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 29
- `take()` (blatann.services.ble\_data\_types.BleDataStream method), 141
- `take_all()` (blatann.services.ble\_data\_types.BleDataStream method), 141
- `tds_control_point` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- `telephone_bearer` (blatann.bt\_sig.uuids.ServiceUuid attribute), 33
- `temperature` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- `temperature_8` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `temperature_8_in_a_period_of_day` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `temperature_8_statistics` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `temperature_celsius` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- `temperature_fahrenheit` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 45
- `temperature_measurement` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `temperature_range` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `temperature_statistics` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `temperature_type` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `termination_reason` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `TesterType` (class in blatann.services.glucose.data\_types), 131
- `then()` (blatann.waitables.event\_waitable.EventWaitable method), 148
- `then()` (blatann.waitables.waitable.EmptyWaitable method), 150
- `then()` (blatann.waitables.waitable.Waitable method), 149
- `thermal_conductivity_watt_per_metre_kelvin` (blatann.bt\_sig.assigned\_numbers.Units attribute), 28
- `thermodynamic_temperature_degree_celsius` (blatann.bt\_sig.assigned\_numbers.Units attribute), 28
- `thermodynamic_temperature_degree_fahrenheit` (blatann.bt\_sig.assigned\_numbers.Units attribute), 28
- `thermodynamic_temperature_kelvin` (blatann.bt\_sig.assigned\_numbers.Units attribute), 28
- `thermometer` (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 29
- `thermometer_ear` (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 29
- `three_zone_heart_rate_limits` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `thursday` (blatann.services.ble\_data\_types.DayOfWeek attribute), 145
- `time_accuracy` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `time_broadcast` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `time_change_log_data` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `time_day` (blatann.bt\_sig.assigned\_numbers.Units attribute), 28
- `time_decihour_8` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `time_exponential_8` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46
- `time_fault` (blatann.services.glucose.data\_types.GlucoseFeatureType attribute), 133
- `time_fault` (blatann.services.glucose.data\_types.SensorStatusType attribute), 133
- `time_hour` (blatann.bt\_sig.assigned\_numbers.Units attribute), 28
- `time_hour_24` (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46

attribute), 46  
 time\_millisecond\_24 (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46  
 time\_minute (blatann.bt\_sig.assigned\_numbers.Units attribute), 28  
 time\_month (blatann.bt\_sig.assigned\_numbers.Units attribute), 28  
 time\_second (blatann.bt\_sig.assigned\_numbers.Units attribute), 28  
 time\_second\_16 (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46  
 time\_second\_8 (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46  
 time\_source (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46  
 time\_trigger\_setting (blatann.bt\_sig.uuids.DescriptorUuid attribute), 31  
 time\_update\_control\_point (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46  
 time\_update\_state (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46  
 time\_with\_dst (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46  
 time\_year (blatann.bt\_sig.assigned\_numbers.Units attribute), 28  
 time\_zone (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46  
 time\_zone\_change (blatann.services.current\_time.data\_types.AdjustmentReasonType attribute), 123  
 TimeAccuracy (class in blatann.services.current\_time.data\_types), 124  
 TIMED\_OUT (blatann.event\_args.GattOperationCompleteReason attribute), 154  
 timeout (blatann.nrf.nrf\_types.enums.BLEGapSecStatus attribute), 105  
 timeout (blatann.nrf.nrf\_types.enums.NrfError attribute), 102  
 timeout\_ms (blatann.gap.gap\_types.ActiveConnectionParameters property), 70  
 TimeoutError, 159  
 TimeRange (class in blatann.nrf.nrf\_types.gap), 108  
 TimeSource (class in blatann.services.current\_time.data\_types), 123  
 tmap\_role (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 46  
 tmas (blatann.bt\_sig.uuids.ServiceUuid attribute), 33  
 to\_ble\_adv\_data() (blatann.gap.advertise\_data.AdvertisingData method), 62  
 to\_buffer() (blatann.gatt.SubscriptionState class method), 77  
 to\_bytes() (blatann.gap.advertise\_data.AdvertisingData method), 62  
 to\_c() (blatann.nrf.nrf\_types.config.BleEnableOpt method), 98  
 to\_c() (blatann.nrf.nrf\_types.config.BleOptGapAuthPayloadTimeout method), 100  
 to\_c() (blatann.nrf.nrf\_types.config.BleOptGapChannelMap method), 99  
 to\_c() (blatann.nrf.nrf\_types.config.BleOptGapLocalConnLatency method), 99  
 to\_c() (blatann.nrf.nrf\_types.config.BleOptGapPaskey method), 99  
 to\_c() (blatann.nrf.nrf\_types.config.BleOptGapSlaveLatencyDisable method), 100  
 to\_c() (blatann.nrf.nrf\_types.config.BleOption method), 98  
 to\_c() (blatann.nrf.nrf\_types.config.BleOptPaLna method), 99  
 to\_c() (blatann.nrf.nrf\_types.config.BlePaLnaConfig method), 99  
 to\_c() (blatann.nrf.nrf\_types.gap.BLEAdvData method), 110  
 to\_c() (blatann.nrf.nrf\_types.gap.BLEGapAddr method), 109  
 to\_c() (blatann.nrf.nrf\_types.gap.BLEGapAdvParams method), 108  
 to\_c() (blatann.nrf.nrf\_types.gap.BLEGapConnParams method), 108  
 to\_c() (blatann.nrf.nrf\_types.gap.BLEGapDataLengthParams method), 110  
 to\_c() (blatann.nrf.nrf\_types.gap.BLEGapPhys method), 110  
 to\_c() (blatann.nrf.nrf\_types.gap.BLEGapPrivacyParams method), 110  
 to\_c() (blatann.nrf.nrf\_types.gap.BLEGapScanParams method), 108  
 to\_c() (blatann.nrf.nrf\_types.gatt.BLEGattCharacteristicProperties method), 111  
 to\_c() (blatann.nrf.nrf\_types.gatt.BLEGattWriteParams method), 111  
 to\_c() (blatann.nrf.nrf\_types.gatt.BleGattEnableParams method), 110  
 to\_c() (blatann.nrf.nrf\_types.gatt.BLEGattExtendedCharacteristicProperties method), 111  
 to\_c() (blatann.nrf.nrf\_types.gatt.BLEGattsAttribute method), 112  
 to\_c() (blatann.nrf.nrf\_types.gatt.BLEGattsAttrMetadata method), 112  
 to\_c() (blatann.nrf.nrf\_types.gatt.BLEGattsAuthorizeParams method), 112  
 to\_c() (blatann.nrf.nrf\_types.gatt.BLEGattsCharHandles



<i>method</i> ), 112	<i>method</i> ), 115
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.gatt.BLEGattsCharMetadata</i> to_dict() ( <i>blatann.nrf.nrf_types.smp.BLEGapSignKey</i> <i>method</i> ), 112	<i>method</i> ), 116
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.gatt.BleGattsEnableParams</i> to_list() ( <i>blatann.nrf.nrf_types.gap.BLEAdvData</i> <i>method</i> ), 112	<i>method</i> ), 110
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.gatt.BLEGattsHvx</i> top ( <i>blatann.bt_sig.assigned_numbers.NamespaceDescriptor</i> <i>method</i> ), 113	<i>attribute</i> ), 25
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.gatt.BLEGattsPresentationFormat</i> track_changed ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 112	<i>attribute</i> ), 46
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.gatt.BLEGattsRwAuthorizeRequest</i> track_duration ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 113	<i>attribute</i> ), 46
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.gatt.BLEGattsValue</i> track_object_type ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 113	<i>attribute</i> ), 46
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.generic.BLEUUID</i> track_position ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 114	<i>attribute</i> ), 46
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.generic.BLEUUIDBase</i> track_segments_object_type ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 113	<i>attribute</i> ), 46
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapDhKey</i> track_title ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 116	<i>attribute</i> ), 46
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapEncryptInfo</i> track_title ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 115	<i>attribute</i> ), 46
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapEncryptKey</i> training_status ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 115	<i>attribute</i> ), 46
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapIdKey</i> transfer_rate_milliliter_per_kilogram_per_minute ( <i>blatann.bt_sig.assigned_numbers.Units</i> <i>method</i> ), 115	<i>attribute</i> ), 29
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapMasterId</i> transport_discovery ( <i>blatann.bt_sig.uuids.ServiceUuid</i> <i>method</i> ), 115	<i>attribute</i> ), 33
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapPublicKey</i> treadmill_data ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 115	<i>attribute</i> ), 47
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapSecKeyDist</i> true_wind_direction ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 114	<i>attribute</i> ), 47
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapSecKeys</i> true_wind_speed ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 116	<i>attribute</i> ), 47
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapSecKeyset</i> try_get_enum() ( <i>blatann.gatt.PresentationFormat</i> <i>method</i> ), 116	<i>static method</i> ), 78
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapSecLevels</i> tuesday ( <i>blatann.services.ble_data_types.DayOfWeek</i> <i>method</i> ), 114	<i>attribute</i> ), 144
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapSecMode</i> two_hour_dst ( <i>blatann.services.current_time.data_types.DaylightSavings</i> <i>method</i> ), 114	<i>attribute</i> ), 123
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapSecParams</i> two_mbps ( <i>blatann.gap.gap_types.Phy</i> <i>method</i> ), 114	<i>attribute</i> ), 69
<code>to_c()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapSignKey</i> two_mbps ( <i>blatann.nrf.nrf_types.enums.BLEGapPhy</i> <i>method</i> ), 116	<i>attribute</i> ), 104
<code>to_dict()</code> ( <i>blatann.gap.bond_db.BondDbEntry</i> two_zone_heart_rate_limit ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> <i>method</i> ), 66	<i>attribute</i> ), 47
<code>to_dict()</code> ( <i>blatann.gap.bond_db.BondingData</i> twobit ( <i>blatann.bt_sig.assigned_numbers.Format</i> <i>method</i> ), 66	<i>attribute</i> ), 47
<code>to_dict()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapEncryptInfo</i> <i>method</i> ), 115	
<code>to_dict()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapEncryptKey</i> <i>method</i> ), 115	
<code>to_dict()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapIdKey</i> <i>method</i> ), 115	
<code>to_dict()</code> ( <i>blatann.nrf.nrf_types.smp.BLEGapMasterId</i> <i>method</i> ), 115	

attribute), 23  
 tx\_power (blatann.bt\_sig.uuids.ServiceUuid attribute), 33  
 tx\_power\_level (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 47  
 tx\_power\_level (blatann.gap.advertise\_data.AdvertisingData.Types attribute), 60  
 tx\_power\_level (blatann.nrf.nrf\_types.gap.BLEAdvData.Types attribute), 109

## U

uint12 (blatann.bt\_sig.assigned\_numbers.Format attribute), 23  
 uint128 (blatann.bt\_sig.assigned\_numbers.Format attribute), 23  
 uint16 (blatann.bt\_sig.assigned\_numbers.Format attribute), 23  
 Uint16 (class in blatann.services.ble\_data\_types), 143  
 uint16\_array\_to\_list() (in module blatann.nrf.nrf\_driver\_types), 119  
 uint24 (blatann.bt\_sig.assigned\_numbers.Format attribute), 23  
 Uint24 (class in blatann.services.ble\_data\_types), 143  
 uint32 (blatann.bt\_sig.assigned\_numbers.Format attribute), 23  
 Uint32 (class in blatann.services.ble\_data\_types), 143  
 Uint40 (class in blatann.services.ble\_data\_types), 143  
 uint48 (blatann.bt\_sig.assigned\_numbers.Format attribute), 23  
 Uint48 (class in blatann.services.ble\_data\_types), 143  
 Uint56 (class in blatann.services.ble\_data\_types), 143  
 uint64 (blatann.bt\_sig.assigned\_numbers.Format attribute), 23  
 Uint64 (class in blatann.services.ble\_data\_types), 143  
 uint8 (blatann.bt\_sig.assigned\_numbers.Format attribute), 23  
 Uint8 (class in blatann.services.ble\_data\_types), 142  
 uint8\_array\_to\_list() (in module blatann.nrf.nrf\_driver\_types), 119  
 uncertainty (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 47  
 under\_stress (blatann.services.glucose.data\_types.HealthStatus attribute), 132  
 undetermined\_plasma (blatann.services.glucose.data\_types.GlucoseType attribute), 130  
 undetermined\_whole\_blood (blatann.services.glucose.data\_types.GlucoseType attribute), 130  
 unitless (blatann.bt\_sig.assigned\_numbers.Units attribute), 25

units (blatann.nrf.nrf\_types.gap.TimeRange property), 108  
 Units (class in blatann.bt\_sig.assigned\_numbers), 25  
 units\_to\_msec() (in module blatann.nrf.nrf\_driver\_types), 119  
 unknown (blatann.bt\_sig.assigned\_numbers.Appearance attribute), 29  
 unknown (blatann.bt\_sig.assigned\_numbers.Namespace attribute), 24  
 unknown (blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor attribute), 25  
 unknown (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106  
 unknown (blatann.nrf.nrf\_types.generic.BLEUUID.Standard attribute), 113  
 unknown (blatann.services.ble\_data\_types.DayOfWeek attribute), 144  
 unknown (blatann.services.current\_time.data\_types.DaylightSavingsTimeOffset attribute), 123  
 unknown (blatann.services.current\_time.data\_types.TimeAccuracy attribute), 124  
 unknown (blatann.services.current\_time.data\_types.TimeSource attribute), 123  
 unknown (blatann.services.glucose.data\_types.SampleLocation attribute), 131  
 unknown\_bt\_le\_command (blatann.nrf.nrf\_types.enums.BLEHci attribute), 100  
 unknown\_connection\_identifier (blatann.nrf.nrf\_types.enums.BLEHci attribute), 101  
 unlikely\_error (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106  
 unread\_alert\_status (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 47  
 UnsignedIntegerBase (class in blatann.services.ble\_data\_types), 142  
 unspecified (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 47  
 unspecified (blatann.nrf.nrf\_types.enums.BLEGapSecStatus attribute), 105  
 unspecified\_error (blatann.nrf.nrf\_types.enums.BLEHci attribute), 101  
 unsubscribe() (blatann.gatt.gattc.GattcCharacteristic method), 80  
 unsupported\_group\_type (blatann.nrf.nrf\_types.enums.BLEGattStatusCode attribute), 106  
 unsupported\_remote\_feature (blatann.nrf.nrf\_types.enums.BLEHci attribute), 101

- unused (blatann.nrf.nrf\_types.enums.BLEGattExecWriteFlags attribute), 107
- update() (blatann.gap.advertise\_data.ScanReport method), 62
- update() (blatann.gap.advertise\_data.ScanReportCollection method), 63
- update() (blatann.gap.bond\_db.BondDatabase method), 66
- update() (blatann.gap.default\_bond\_db.DefaultBondDatabase method), 69
- update() (blatann.gap.generic\_access\_service.GenericAccessService method), 71
- update() (blatann.gap.scanning.ScanParameters method), 71
- update() (blatann.gatt.gattc\_attribute.GattcAttribute method), 83
- update\_connection\_parameters() (blatann.peer.Peer method), 162
- update\_data\_length() (blatann.peer.Peer method), 163
- update\_phy() (blatann.peer.Peer method), 163
- upper (blatann.bt\_sig.assigned\_numbers.NamespaceDescriptor attribute), 25
- uri (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 47
- uri (blatann.gap.advertise\_data.AdvertisingData.Types attribute), 61
- uri (blatann.nrf.nrf\_types.gap.BLEAdvData.Types attribute), 110
- usb\_vendor (blatann.services.device\_info.data\_types.PnpVendorSource attribute), 128
- use\_debug\_leesc\_key() (blatann.gap.smp.SecurityManager method), 75
- user\_control\_point (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 47
- user\_data (blatann.bt\_sig.uuids.ServiceUuid attribute), 34
- user\_description (blatann.bt\_sig.uuids.DescriptorUuid attribute), 31
- user\_description (blatann.gatt.gatts.GattsCharacteristic property), 86
- user\_facing\_time (blatann.services.glucose.racp.FilterType attribute), 138
- user\_index (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 47
- user\_rejected (blatann.event\_args.PairingRejectedReason attribute), 156
- utf16s (blatann.bt\_sig.assigned\_numbers.Format attribute), 24
- uuid (blatann.gatt.Attribute property), 77
- Uuid (class in blatann.uuid), 165
- Uuid128 (class in blatann.uuid), 165
- uuid16 (blatann.uuid.Uuid128 property), 165
- Uuid16 (class in blatann.uuid), 165
- uuid\_base (blatann.uuid.Uuid128 property), 165
- uuid\_index (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 47
- ## V
- valid\_range (blatann.bt\_sig.uuids.DescriptorUuid attribute), 31
- validate() (blatann.gap.scanning.ScanParameters method), 71
- validate() (blatann.nrf.nrf\_types.gap.BLEGapConnParams method), 108
- validate() (blatann.nrf.nrf\_types.gap.TimeRange method), 108
- value (blatann.gatt.Attribute property), 77
- value (blatann.gatt.gattc.GattcCharacteristic property), 78
- value (blatann.gatt.gatts.GattsCharacteristic property), 86
- value\_attribute (blatann.gatt.gattc.GattcCharacteristic property), 78
- value\_trigger\_setting (blatann.bt\_sig.uuids.DescriptorUuid attribute), 31
- velocity\_kilometer\_per\_minute (blatann.bt\_sig.assigned\_numbers.Units attribute), 29
- velocity\_kilometre\_per\_hour (blatann.bt\_sig.assigned\_numbers.Units attribute), 29
- velocity\_knot (blatann.bt\_sig.assigned\_numbers.Units attribute), 29
- velocity\_metres\_per\_second (blatann.bt\_sig.assigned\_numbers.Units attribute), 29
- velocity\_mile\_per\_hour (blatann.bt\_sig.assigned\_numbers.Units attribute), 29
- venous\_plasma (blatann.services.glucose.data\_types.GlucoseType attribute), 130
- venous\_whole\_blood (blatann.services.glucose.data\_types.GlucoseType attribute), 130
- vo2\_max (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 47
- voltage (blatann.bt\_sig.uuids.CharacteristicUuid attribute), 47

<code>voltage_specification</code>	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47	<code>wait_for_user_stop()</code> (in module <i>blatann.examples.broadcaster</i> ), 49
<code>voltage_statistics</code>	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47	<code>Waitable</code> (class in <i>blatann.waitables.waitable</i> ), 149
<code>volume_control</code>	( <i>blatann.bt_sig.uuids.ServiceUuid</i> attribute), 34	<code>watch</code> ( <i>blatann.bt_sig.assigned_numbers.Appearance</i> attribute), 29
<code>volume_control_point</code>	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47	<code>wavenumber_reciprocal_metre</code> ( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 29
<code>volume_cubic_metres</code>	( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 29	<code>wednesday</code> ( <i>blatann.services.ble_data_types.DayOfWeek</i> attribute), 144
<code>volume_flags</code>	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47	<code>weight</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47
<code>volume_flow</code>	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47	<code>weight_measurement</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47
<code>volume_flow_litre_per_second</code>	( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 29	<code>weight_scale</code> ( <i>blatann.bt_sig.assigned_numbers.Appearance</i> attribute), 30
<code>volume_litre</code>	( <i>blatann.bt_sig.assigned_numbers.Units</i> attribute), 29	<code>weight_scale</code> ( <i>blatann.bt_sig.uuids.ServiceUuid</i> attribute), 34
<code>volume_offset_control</code>	( <i>blatann.bt_sig.uuids.ServiceUuid</i> attribute), 34	<code>weight_scale_feature</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47
<code>volume_offset_control_point</code>	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47	<code>wind_chill</code> ( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47
<code>volume_offset_state</code>	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47	<code>within_range_inclusive</code> ( <i>blatann.services.glucose.racp.RacpOperator</i> attribute), 138
<code>volume_state</code>	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47	<code>writable</code> ( <i>blatann.gatt.gattc.GattcCharacteristic</i> property), 79
<b>W</b>		<code>writable_without_response</code> ( <i>blatann.gatt.gattc.GattcCharacteristic</i> property), 79
<code>waist_circumference</code>	( <i>blatann.bt_sig.uuids.CharacteristicUuid</i> attribute), 47	<code>write()</code> ( <i>blatann.gatt.gattc.GattcCharacteristic</i> method), 80
<code>wait()</code> ( <i>blatann.waitables.connection_waitable.ClientConnectionWaitable</i> method), 147		<code>write()</code> ( <i>blatann.gatt.gattc_attribute.GattcAttribute</i> method), 83
<code>wait()</code> ( <i>blatann.waitables.connection_waitable.ConnectionWaitable</i> method), 147		<code>write()</code> ( <i>blatann.gatt.managers.GattcOperationManager</i> method), 90
<code>wait()</code> ( <i>blatann.waitables.connection_waitable.PeripheralConnectionWaitable</i> method), 147		<code>write()</code> ( <i>blatann.gatt.writer.GattcWriter</i> method), 91
<code>wait()</code> ( <i>blatann.waitables.event_waitable.EventWaitable</i> method), 148		<code>write()</code> ( <i>blatann.services.nordic_uart.service.NordicUartClient</i> method), 141
<code>wait()</code> ( <i>blatann.waitables.scan_waitable.ScanFinishedWaitable</i> method), 149		<code>write()</code> ( <i>blatann.services.nordic_uart.service.NordicUartServer</i> method), 140
<code>wait()</code> ( <i>blatann.waitables.waitable.EmptyWaitable</i> method), 150		<code>write_cmd</code> ( <i>blatann.nrf.nrf_types.enums.BLEGattsWriteOperation</i> attribute), 107
<code>wait()</code> ( <i>blatann.waitables.waitable.Waitable</i> method), 149		<code>write_cmd</code> ( <i>blatann.nrf.nrf_types.enums.BLEGattWriteOperation</i> attribute), 105
		<code>write_not_permitted</code> ( <i>blatann.nrf.nrf_types.enums.BLEGattStatusCode</i> attribute), 106
		<code>write_req</code> ( <i>blatann.nrf.nrf_types.enums.BLEGattsWriteOperation</i> attribute), 107
		<code>write_req</code> ( <i>blatann.nrf.nrf_types.enums.BLEGattWriteOperation</i> attribute), 105

`write_without_response()` (*blatann.gatt.gattc.GattcCharacteristic* method), [81](#)

`WriteCompleteEventArgs` (*class in blatann.event\_args*), [157](#)

`WriteEventArgs` (*class in blatann.event\_args*), [156](#)

## X

`x_trans_key_disallowed` (*blatann.nrf.nrf\_types.enums.BLEGapSecStatus* attribute), [105](#)