bluetooth-numbers Documentation

Release 1.1.0.post1.dev1+g082e59c

Koen Vervloesem

Feb 20, 2023

CONTENTS

Python package with a wide set of numbers related to Bluetooth

This project offers a Python package with a wide set of numbers related to Bluetooth, so Python projects can easily use these numbers. The goal of this project is to provide a shared resource so various Python projects that deal with Bluetooth don't have to replicate this effort by rolling their own database and keeping it updated.

The following sources are used:

- Nordic Semiconductor's Bluetooth Numbers Database for Company IDs, Service UUIDs, Characteristic UUIDs and Descriptor UUIDs
- Bluetooth Assigned Numbers for SDO Service UUIDs and Member Service UUIDs
- The IEEE database of OUIs for prefixes of Bluetooth addresses

CHAPTER

ONE

CONTENTS

1.1 Installation and usage

1.1.1 Installation

You can install bluetooth-numbers as a package from PyPI with pip:

```
pip install bluetooth-numbers
```

1.1.2 Usage

Get the description of a company ID:

```
>>> from bluetooth_numbers import company
>>> company[0x0499]
'Ruuvi Innovations Ltd.'
```

Get the description of a service UUID:

```
>>> from bluetooth_numbers import service
>>> from uuid import UUID
>>> service[0x180F]
'Battery Service'
>>> service[UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")]
'Nordic UART Service'
```

Get the description of a characteristic UUID:

```
>>> from bluetooth_numbers import characteristic
>>> from uuid import UUID
>>> characteristic[0x2A37]
'Heart Rate Measurement'
>>> characteristic[UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E")]
'UART RX Characteristic'
```

Get the description of a descriptor UUID:

```
>>> from bluetooth_numbers import descriptor
>>> descriptor[0x2901]
'Characteristic User Descriptor'
```

Get the description of an OUI:

```
>>> from bluetooth_numbers import oui
>>> oui["58:2D:34"]
'Qingping Electronics (Suzhou) Co., Ltd'
```

See the module reference for complete documentation.

1.2 Contributing

Welcome to `bluetooth-numbers` contributor's guide.

This document focuses on getting any potential contributor familiarized with the development processes, but other kinds of contributions are also appreciated.

If you are new to using git or have never collaborated in a project previously, please have a look at contribution-guide.org. Other resources are also listed in the excellent guide created by FreeCodeCamp[1].

Please notice, all users and contributors are expected to be **open, considerate, reasonable, and respectful**. When in doubt, Python Software Foundation's Code of Conduct is a good reference in terms of behavior guidelines.

1.2.1 Issue Reports

If you experience bugs or general issues with `bluetooth-numbers`, please have a look on the issue tracker. If you don't see anything useful there, please feel free to fire an issue report.

Tip: Please don't forget to include the closed issues in your search. Sometimes a solution was already reported, and the problem is considered **solved**.

New issue reports should include information about your programming environment (e.g., operating system, Python version) and steps to reproduce the problem. Please try also to simplify the reproduction steps to a very minimal example that still illustrates the problem you are facing. By removing other factors, you help us to identify the root cause of the issue.

1.2.2 Documentation Improvements

You can help improve `bluetooth-numbers` docs by making them more readable and coherent, or by adding missing information and correcting mistakes.

`bluetooth-numbers` documentation uses Sphinx as its main documentation compiler. This means that the docs are kept in the same repository as the project code, and that any documentation update is done in the same way was a code contribution.

The documentation is written in the reStructuredText markup language.

> **Tip:** Please notice that the GitHub web interface provides a quick way of propose changes in `bluetooth-numbers`'s files. While this mechanism can be tricky for normal code contributions, it works perfectly fine for contributing to the docs, and can be quite handy.

[1] Even though, these resources focus on open source projects and communities, the general ideas behind collaborating with other developers to collectively create software are general and can be applied to all sorts of environments, including private companies and proprietary code bases.

> If you are interested in trying this method out, please navigate to the `docs` folder in the source repository, find which file you would like to propose changes and click in the little pencil icon at the top, to open GitHub's code editor. Once you finish editing the file, please write a message in the form at the bottom of the page describing which changes have you made and what are the motivations behind them and submit your proposal.

When working on documentation changes in your local machine, you can compile them using `tox`:

```
tox -e docs
```

and use Python's built-in web server for a preview in your web browser (`http://localhost:8000`):

```
python3 -m http.server --directory 'docs/_build/html'
```

1.2.3 Code Contributions

The goal of this project is to provide a shared resource so various Python projects that deal with Bluetooth don't have to replicate this effort by rolling their own database and keeping it updated. We welcome any contribution that helps reaching this goal.

That said, we try to contribute as much as possible to the upstream sources of the data in this package. So, if you want to extend the company IDs, service UUIDs, characteristic UUIDs or descriptor UUIDs, or if you want to correct some data, we ask you to contribute this to Nordic Semiconductor's Bluetooth Numbers Database if possible.

Submit an issue

Before you work on any non-trivial code contribution it's best to first create a report in the issue tracker to start a discussion on the subject. This often provides additional considerations and avoids unnecessary work.

Create an environment

Before you start coding, we recommend creating an isolated virtual environment to avoid any problems with your installed Python packages. This can easily be done via either `virtualenv`:

```
virtualenv <PATH TO VENV>
source <PATH TO VENV>/bin/activate
```

or Miniconda:

```
conda create -n bluetooth-numbers python=3 six virtualenv pytest pytest-cov
conda activate bluetooth-numbers
```

Clone the repository

1. Create an user account on GitHub if you do not already have one.
2. Fork the project repository: click on the *Fork* button near the top of the page. This creates a copy of the code under your account on GitHub.
3. Clone this copy to your local disk:

```
git clone git@github.com:YourLogin/bluetooth-numbers.git
cd bluetooth-numbers
```

4. You should run:

```
pip install -U pip setuptools -e .
```

   to be able to import the package under development in the Python REPL.

5. Install `pre-commit`:

```
pip install pre-commit
pre-commit install
```

   `bluetooth-numbers` comes with a lot of hooks configured to automatically help the developer to check the code being written.

Implement your changes

1. Create a branch to hold your changes:

```
git checkout -b my-feature
```

   and start making changes. Never work on the main branch!

2. Start your work on this branch. Don't forget to add docstrings to new functions, modules and classes, especially if they are part of public APIs.
3. Add yourself to the list of contributors in `AUTHORS.rst`.
4. When you're done editing, do:

```
git add <MODIFIED FILES>
git commit
```

   to record your changes in git.

   Please make sure to see the validation messages from `pre-commit` and fix any eventual issues. This should automatically use flake8/black to check/fix the code style in a way that is compatible with the project.

Important: Don't forget to add unit tests and documentation in case your contribution adds an additional feature and is not just a bugfix.

Moreover, writing a descriptive commit message is highly recommended. In case of doubt, you can check the commit history with:

```
git log --graph --decorate --pretty=oneline --abbrev-commit --all
```

to look for recurring communication patterns.

5. Please check that your changes don't break any unit tests with:

```
tox
```

(after having installed `tox` with `pip install tox` or `pipx`).

You can also use `tox` to run several other pre-configured tasks in the repository. Try `tox -av` to see a list of the available checks.

Submit your contribution

1. If everything works fine, push your local branch to GitHub with:

```
git push -u origin my-feature
```

2. Go to the web page of your fork and click "Create pull request" to send your changes for review.

Find more detailed information in creating a PR. You might also want to open the PR as a draft first and mark it as ready for review after the feedbacks from the continuous integration (CI) system or any required fixes.

Troubleshooting

The following tips can be used when facing problems to build or test the package:

1. Make sure to fetch all the tags from the upstream repository. The command `git describe --abbrev=0 --tags` should return the version you are expecting. If you are trying to run CI scripts in a fork repository, make sure to push all the tags. You can also try to remove all the egg files or the complete egg folder, i.e., `.eggs`, as well as the `*.egg-info` folders in the `src` folder or potentially in the root of your project.
2. Sometimes `tox` misses out when new dependencies are added, especially to `setup.cfg` and `docs/requirements.txt`. If you find any problems with missing dependencies when running a command with `tox`, try to recreate the `tox` environment using the `-r` flag. For example, instead of:

```
tox -e docs
```

Try running:

```
tox -r -e docs
```

3. Make sure to have a reliable `tox` installation that uses the correct Python version (e.g., 3.7+). When in doubt you can run:

```
tox --version
# OR
which tox
```

If you have trouble and are seeing weird errors upon running `tox`, you can also try to create a dedicated virtual environment with a `tox` binary freshly installed. For example:

```
virtualenv .venv
source .venv/bin/activate
.venv/bin/pip install tox
.venv/bin/tox -e all
```

4. Pytest can drop you in an interactive session in the case an error occurs. In order to do that you need to pass a `--pdb` option (for example by running `tox -- -k <NAME OF THE FALLING TEST> --pdb`). You can also setup breakpoints manually instead of using the `--pdb` option.

1.2.4 Maintainer tasks

Updating data

1. Run `git submodule update --remote` to update the Bluetooth Numbers Database to the newest version.
2. Enter the `data` directory and run `wget https://standards-oui.ieee.org/oui/oui.txt` to download the newest OUIs.
3. Download the latest Assigned Numbers document and add the new values to `data/member_service_uuids.json` and/or `data/sdo_service_uuids.json`.
4. Run `pre-commit run generate-modules --hook-stage manual --all-files` to generate the project's modules from the newest data.
5. Run `pre-commit run --all-files` to clean up the generated modules.

Releases

If you are part of the group of maintainers and have correct user permissions on PyPI, the following steps can be used to release a new version for `bluetooth-numbers`:

1. Make sure all unit tests are successful by running `tox`.
2. Tag the current commit on the main branch with a release tag, e.g., `v1.2.3`.
3. Push the new tag to the upstream repository, e.g., `git push upstream v1.2.3`
4. Clean up the `dist` and `build` folders with `tox -e clean` (or `rm -rf dist build`) to avoid confusion with old builds and Sphinx docs.
5. Run `tox -e build` and check that the files in `dist` have the correct version (no `.dirty` or git hash) according to the git tag. Also check the sizes of the distributions, if they are too big (e.g., > 500KB), unwanted clutter may have been accidentally included.
6. Run `tox -e publish -- --repository pypi` and check that everything was uploaded to PyPI correctly.

1.3 License

1.4 Contributors

- Koen Vervloesem

1.5 Changelog

1.5.1 Version 1.1.0 (2023-02-20)

This is a small feature release. Apart from the updated data, there's a new function *bluetooth_numbers.utils.is_standard_uuid128()* that checks whether a 128-bit Bluetooth UUID is a standard UUID.

What's Changed

- Fix publish job in CI: it needs the test job by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/24
- Update OUI badge to 2023-01-25 by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/25
- Autoupdate pre-commit by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/26
- Add function is_standard_uuid128 to utils by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/27
- Update Bluetooth Numbers Database to commit e1683ef by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/28
- Update OUI database to 2023-02-20 by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/29
- Update member service UUIDs to 2023-02-17 by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/30

- Add info about updating data to CONTRIBUTING docs by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/31

1.5.2 Version 1.0.1 (2023-01-25)

This is a bugfix release, mostly with updated data:

- The OUI database has been updated to 2023-01-25.
- The Bluetooth Numbers Database has been updated with some fixes. Upstream PRs: https://github.com/NordicSemiconductor/bluetooth-numbers-database/pull/93 and https://github.com/NordicSemiconductor/bluetooth-numbers-database/pull/96.

What's Changed

- CI: Run publish job in Python 3.11 by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/8
- Fix docstrings for correct rendering by Sphinx by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/9
- Specify Python version with pipx run by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/11
- Add modern typing by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/10
- Check tests for style by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/12
- Enable extra Flake8 plugins by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/13
- Adds and updates pre-commit hooks by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/14
- Improve index, installation and usage pages of documentation by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/15
- Add badges with amount of Bluetooth numbers to README by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/16
- Update to PyScaffold v4.4 project features by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/18
- Change documentation theme to furo by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/19
- Documentation updates by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/20
- Update Bluetooth Numbers Database to commit d05e669 by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/21
- Update OUI database to 2023-01-25 by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/22

1.5.3 Version 1.0.0 (2022-12-22)

This is a major release with some breaking changes.

Whereas in previous versions you did:

```
from bluetooth_numbers.companies import company
```

This is now:

```
from bluetooth_numbers import company
```

The OUIs and CICs now also use their own dict-like class, just like the services, characteristics and descriptions already did.

All searches for numbers now raise package-specific exceptions when something's wrong, for instance for invalid or unknown values.

Look at the API documentation for all these changes.

What's Changed

- Documentation improvements with docstrings by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/1
- Run doctests in CI to make sure examples in the documentation work by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/2
- Add package data for minimum Python version and keywords by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/3
- Run mypy in pre-commit hook by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/4
- Add custom exceptions for this package by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/5
- Change public API for easier importing by @koenvervloesem in https://github.com/koenvervloesem/bluetooth-numbers/pull/6

1.5.4 Version 0.2.1 (2022-12-20)

This bugfix release updates the Bluetooth Numbers Database to commit 3d0f452 (December 20 2022). This fixes some issues with Philips Hue UUIDs. Upstream PR: NordicSemiconductor/bluetooth-numbers-database#94.

1.5.5 Version 0.2.0 (2022-12-19)

- Adds SDO service UUIDs.
- Adds member service UUIDs.

Both types of UUIDs are taken from the Bluetooth Assigned Numbers document from 2022-12-15.

1.5.6 Version 0.1.3 (2022-12-18)

- Adds typing to company dict.
- Tracks bluetooth-numbers-database @ 4a5f38a.

1.5.7 Version 0.1.2 (2022-07-05)

Updates company IDs, services, characteristics and descriptors. This tracks bluetooth-numbers-database @ 2178b94 (July 5 2022).

1.5.8 Version 0.1.1 (2022-07-01)

Initial release

1.6 bluetooth_numbers

1.6.1 bluetooth_numbers package

bluetooth-numbers: Python package with a wide set of numbers related to Bluetooth.

This project offers a Python package with company IDs, service, characteristic and descriptor UUIDs and OUIs, so Python projects can easily use these numbers.

Get the description of a company ID:

```
>>> from bluetooth_numbers import company
>>> company[0x0499]
'Ruuvi Innovations Ltd.'
```

Get the description of a service UUID:

```
>>> from bluetooth_numbers import service
>>> from uuid import UUID
>>> service[0x180F]
'Battery Service'
>>> service[UUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")]
'Nordic UART Service'
```

Get the description of a characteristic UUID:

```
>>> from bluetooth_numbers import characteristic
>>> from uuid import UUID
>>> characteristic[0x2A37]
'Heart Rate Measurement'
>>> characteristic[UUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E")]
'UART RX Characteristic'
```

Get the description of a descriptor UUID:

```
>>> from bluetooth_numbers import descriptor
>>> descriptor[0x2901]
'Characteristic User Descriptor'
```

Get the description of an OUI:

```
>>> from bluetooth_numbers import oui
>>> oui["58:2D:34"]
'Qingping Electronics (Suzhou) Co., Ltd'
```

Submodules

bluetooth_numbers.dicts module

Module with specialized dictionary classes for UUIDs, CICs and OUIs.

class bluetooth_numbers.dicts.**CICDict**

Bases: Dict[int, str]

Dictionary class to hold 16-bit company codes and their names.

You can use this class as a dict with the following differences:

- If you check for a key that doesn't exist, this raises an *UnknownCICError*.
- If you check for a key that isn't a 16-bit unsigned integer, this raises a *No16BitIntegerError*.

Examples

```
>>> from bluetooth_numbers import company
>>> company[0x004C]
'Apple, Inc.'
>>> company[-1]
Traceback (most recent call last):
bluetooth_numbers.exceptions.No16BitIntegerError: -1
>>> company[65534]
Traceback (most recent call last):
bluetooth_numbers.exceptions.UnknownCICError: 65534
```

class bluetooth_numbers.dicts.**OUIDict**

Bases: Dict[str, str]

Dictionary class to hold OUIs and their names.

You can use this class as a dict with the following differences:

- You can check for an OUI in the formats "xx:yy:zz", "xx-yy-zz" or "xxyyzz". Both lowercase and uppercase letters are supported.
- If you check for a key that doesn't exist, this raises an *UnknownOUIError*.
- If you check for a key that doesn't have one of the supported formats, this raises a *WrongOUIFormatError*.

Examples

```
>>> from bluetooth_numbers import oui
>>> oui["98:E7:43"]
'Dell Inc.'
>>> oui["c4-29-96"]
'Signify B.V.'
>>> oui["A44519"]
'Xiaomi Communications Co Ltd'
>>> oui["FOOBAR"]
Traceback (most recent call last):
bluetooth_numbers.exceptions.WrongOUIFormatError: 'FOOBAR'
>>> oui["AB:CD:EF"]
Traceback (most recent call last):
bluetooth_numbers.exceptions.UnknownOUIError: AB:CD:EF
```

class bluetooth_numbers.dicts.**UUIDDict**

Bases: Dict[Union[UUID, int], str]

Dictionary class to hold 16-bit and 128-bit standard UUID keys and descriptions.

You can use this class as a dict for Bluetooth UUIDs, with the following differences:

- If you check for a 128-bit standard UUID and this UUID doesn't exist in the dictionary, it will check for the corresponding 16-bit UUID.
- If you check for a UUID that doesn't exist, this raises an *UnknownUUIDError*.
- If you check for a key that isn't a 16-bit unsigned integer, this raises a *No16BitIntegerError*.

Examples

```
>>> from bluetooth_numbers import service
>>> from uuid import UUID
>>> service[UUID("0000180F-0000-1000-8000-00805F9B34FB")]
'Battery Service'
>>> service[0x180F]
'Battery Service'
>>> service[0]
Traceback (most recent call last):
bluetooth_numbers.exceptions.UnknownUUIDError: 0
>>> service[6.5]
Traceback (most recent call last):
bluetooth_numbers.exceptions.No16BitIntegerError: 6.5
```

bluetooth_numbers.exceptions module

Module with exceptions raised by this library.

exception bluetooth_numbers.exceptions.**BluetoothNumbersError**

Bases: Exception

Base class for all exceptions raised by this library.

exception bluetooth_numbers.exceptions.**No16BitIntegerError**

Bases: *BluetoothNumbersError*

Exception raised when an integer is not a 16-bit number.

exception bluetooth_numbers.exceptions.**NonStandardUUIDError**

Bases: *BluetoothNumbersError*

Exception raised when a 128-bit UUID is not a standard Bluetooth UUID.

exception bluetooth_numbers.exceptions.**UnknownCICError**

Bases: *BluetoothNumbersError*

Exception raised when a CIC is not known.

exception bluetooth_numbers.exceptions.**UnknownOUIError**

Bases: *BluetoothNumbersError*

Exception raised when an OUI is not known.

exception bluetooth_numbers.exceptions.**UnknownUUIDError**

Bases: *BluetoothNumbersError*

Exception raised when a UUID is not known.

exception bluetooth_numbers.exceptions.**WrongOUIFormatError**

Bases: *BluetoothNumbersError*

Exception raised when a string isn't a supported format for an OUI.

bluetooth_numbers.utils module

Module with utility functions for Bluetooth numbers.

bluetooth_numbers.utils.**BASE_UUID: UUID = UUID('00000000-0000-1000-8000-00805f9b34fb')**

Base UUID defined by the Bluetooth SIG.

bluetooth_numbers.utils.**is_normalized_oui**(*oui: str*) → bool

Check whether the argument is a normalized OUI.

A normalized OUI is a string with format "XX:YY:ZZ" where XX, YY and ZZ are uppercase hexadecimal digits.

Parameters
oui (*str*) – The string to check.

Returns
`True` if *oui* is a normalized OUI, `False` otherwise.

Return type
bool

Examples

```
>>> from bluetooth_numbers.utils import is_normalized_oui
>>> is_normalized_oui("98-e7-43")
False
>>> is_normalized_oui("98:E7:43")
True
>>> is_normalized_oui("FOOBAR")
False
```

bluetooth_numbers.utils.**is_standard_uuid128**(*uuid128: UUID*) → bool

Check whether a 128-bit Bluetooth UUID is a standard UUID.

Parameters
uuid128 (*UUID*) – A 128-bit Bluetooth UUID.

Returns
`True` if *uuid128* is a standard Bluetooth UUID, `False` otherwise.

Return type
bool

Examples

```
>>> from bluetooth_numbers.utils import is_standard_uuid128
>>> from uuid import UUID
>>> is_standard_uuid128(UUID('00001800-0000-1000-8000-00805f9b34fb'))
True
>>> is_standard_uuid128(UUID('bfc46884-ea75-416b-8154-29c5d0b0a087'))
False
```

New in version 1.1.0.

bluetooth_numbers.utils.**is_uint16**(*number: int*) → bool

Check whether a number is a 16-bit unsigned integer.

Parameters
number (*int*) – The number to check.

Returns
`True` if *number* is a 16-bit unsigned integer, `False` otherwise.

Return type
bool

Examples

```
>>> from bluetooth_numbers.utils import is_uint16
>>> is_uint16(0x1800)
True
>>> is_uint16(-1)
False
```

bluetooth_numbers.utils.**normalize_oui**(*oui: str*) → str

Normalize an OUI.

This changes the letters in decimal digits to uppercase and places a colon between the OUI's bytes.

Parameters
oui (*str*) – The OUI to normalize.

Raises
WrongOUIFormatError – If *oui* doesn't have the right format.

Returns
oui as a normalized OUI.

Return type
str

Examples

```
>>> from bluetooth_numbers.utils import normalize_oui
>>> normalize_oui("98-e7-43")
'98:E7:43'
>>> normalize_oui("98e743")
'98:E7:43'
>>> normalize_oui("FOOBAR")
Traceback (most recent call last):
bluetooth_numbers.exceptions.WrongOUIFormatError: 'FOOBAR'
```

bluetooth_numbers.utils.**uint16_to_hex**(*number: int*) → str

Convert a 16-bit UUID or Company ID to a hexadecimal string.

Parameters
number (*int*) – A 16-bit number.

Raises
No16BitIntegerError – If *number* is not an integer from 0 to 65535.

Returns
A hexadecimal string representation of *number*.

Return type
str

Example

```
>>> from bluetooth_numbers.utils import uint16_to_hex
>>> uint16_to_hex(0xFD6F)
'0xfd6f'
```

bluetooth_numbers.utils.**uuid128_to_uuid16**(*uuid128: UUID*) → int

Convert a 128-bit standard Bluetooth UUID to a 16-bit UUID.

Parameters
uuid128 (*UUID*) – A 128-bit Bluetooth UUID.

Raises
NonStandardUUIDError – If *uuid128* is not a 128-bit standard Bluetooth UUID.

Returns
A 16-bit UUID that is the short UUID of *uuid128*.

Return type
int

Example

```
>>> from bluetooth_numbers.utils import uuid128_to_uuid16, uint16_to_hex
>>> from uuid import UUID
>>> uint16_to_hex(uuid128_to_uuid16(UUID('00001800-0000-1000-8000-00805f9b34fb')))
'0x1800'
```

bluetooth_numbers.utils.**uuid16_to_uuid128**(*uuid16: int*) → UUID

Convert a 16-bit UUID to a 128-bit UUID with the Bluetooth base UUID.

Parameters
uuid16 (*int*) – A 16-bit UUID.

Raises
No16BitIntegerError – If *uuid16* is not an integer from 0 to 65535.

Returns
A 128-bit UUID that is the full UUID of *uuid16*.

Return type
UUID

Example

```
>>> from bluetooth_numbers.utils import uuid16_to_uuid128
>>> uuid16_to_uuid128(0x1800)
UUID('00001800-0000-1000-8000-00805f9b34fb')
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX