21.20. `uuid` — UUID objects according to RFC 4122

Source code: Lib/uuid.py

This module provides immutable `UUID` objects (the `UUID` class) and the functions `uuid1()`, `uuid3()`, `uuid4()`, `uuid5()` for generating version 1, 3, 4, and 5 UUIDs as specified in **RFC 4122**.

If all you want is a unique ID, you should probably call `uuid1()` or `uuid4()`. Note that `uuid1()` may compromise privacy since it creates a UUID containing the computer's network address. `uuid4()` creates a random UUID.

class `uuid.`**`UUID`**(*hex=None*, *bytes=None*, *bytes_le=None*, *fields=None*, *int=None*, *version=None*)

Create a UUID from either a string of 32 hexadecimal digits, a string of 16 bytes in big-endian order as the *bytes* argument, a string of 16 bytes in little-endian order as the *bytes_le* argument, a tuple of six integers (32-bit *time_low*, 16-bit *time_mid*, 16-bit *time_hi_version*, 8-bit *clock_seq_hi_variant*, 8-bit *clock_seq_low*, 48-bit *node*) as the *fields* argument, or a single 128-bit integer as the *int* argument. When a string of hex digits is given, curly braces, hyphens, and a URN prefix are all optional. For example, these expressions all yield the same UUID:

```
UUID('{12345678-1234-5678-1234-567812345678}')
UUID('12345678123456781234567812345678')
UUID('urn:uuid:12345678-1234-5678-1234-567812345678')
UUID(bytes=b'\x12\x34\x56\x78'*4)
UUID(bytes_le=b'\x78\x56\x34\x12\x34\x12\x78\x56' +
              b'\x12\x34\x56\x78\x12\x34\x56\x78')
UUID(fields=(0x12345678, 0x1234, 0x5678, 0x12, 0x34, 0x56781234567
UUID(int=0x12345678123456781234567812345678)
```

Exactly one of *hex*, *bytes*, *bytes_le*, *fields*, or *int* must be given. The *version* argument is optional; if given, the resulting UUID will have its variant and version number set according to **RFC 4122**, overriding bits in the given *hex*, *bytes*, *bytes_le*, *fields*, or *int*.

Comparison of UUID objects are made by way of comparing their `UUID.int` attributes. Comparison with a non-UUID object raises a `TypeError`.

`str(uuid)` returns a string in the form `12345678-1234-5678-1234-567812345678` where the 32 hexadecimal digits represent the UUID.

`UUID` instances have these read-only attributes:

`UUID.`**`bytes`**

The UUID as a 16-byte string (containing the six integer fields in big-endian byte order).

`UUID.`**`bytes_le`**

The UUID as a 16-byte string (with *time_low*, *time_mid*, and *time_hi_version* in little-endian byte order).

`UUID.`**`fields`**

A tuple of the six integer fields of the UUID, which are also available as six individual attributes and two derived attributes:

Field	**Meaning**
`time_low`	the first 32 bits of the UUID
`time_mid`	the next 16 bits of the UUID
`time_hi_version`	the next 16 bits of the UUID
`clock_seq_hi_variant`	the next 8 bits of the UUID
`clock_seq_low`	the next 8 bits of the UUID
`node`	the last 48 bits of the UUID
`time`	the 60-bit timestamp
`clock_seq`	the 14-bit sequence number

`UUID.`**`hex`**

The UUID as a 32-character hexadecimal string.

`UUID.`**`int`**

The UUID as a 128-bit integer.

`UUID.`**`urn`**

The UUID as a URN as specified in **RFC 4122**.

`UUID.`**`variant`**

The UUID variant, which determines the internal layout of the UUID. This will be one of the constants `RESERVED_NCS`, `RFC_4122`, `RESERVED_MICROSOFT`, or `RESERVED_FUTURE`.

`UUID.`**`version`**

The UUID version number (1 through 5, meaningful only when the variant is `RFC_4122`).

The `uuid` module defines the following functions:

`uuid.`**`getnode`**()

Get the hardware address as a 48-bit positive integer. The first time this runs, it may launch a separate program, which could be quite slow. If all attempts to obtain the hardware address fail, we choose a random 48-bit number with its eighth bit set to 1 as recommended in **RFC 4122**. “Hardware address” means the MAC address of a network interface, and on a machine with multiple network interfaces the MAC address of any one of them may be returned.

`uuid.`**`uuid1`**(*node=None*, *clock_seq=None*)

Generate a UUID from a host ID, sequence number, and the current time. If *node* is not given, `getnode()` is used to obtain the hardware address. If *clock_seq* is given, it is used as the sequence number; otherwise a random 14-bit sequence number is chosen.

`uuid.`**`uuid3`**(*namespace*, *name*)

Generate a UUID based on the MD5 hash of a namespace identifier (which is a UUID) and a name (which is a string).

`uuid.`**`uuid4`**()

Generate a random UUID.

`uuid.`**`uuid5`**(*namespace*, *name*)

Generate a UUID based on the SHA-1 hash of a namespace identifier (which is a UUID) and a name (which is a string).

The `uuid` module defines the following namespace identifiers for use with `uuid3()` or `uuid5()`.

`uuid.`**`NAMESPACE_DNS`**

When this namespace is specified, the *name* string is a fully-qualified domain name.

`uuid.`**`NAMESPACE_URL`**

When this namespace is specified, the *name* string is a URL.

`uuid.`**`NAMESPACE_OID`**

When this namespace is specified, the *name* string is an ISO OID.

`uuid.`**`NAMESPACE_X500`**

When this namespace is specified, the *name* string is an X.500 DN in DER or a text output format.

The `uuid` module defines the following constants for the possible values of the `variant` attribute:

`uuid.`**`RESERVED_NCS`**
Reserved for NCS compatibility.

`uuid.`**`RFC_4122`**
Specifies the UUID layout given in **RFC 4122**.

`uuid.`**`RESERVED_MICROSOFT`**
Reserved for Microsoft compatibility.

`uuid.`**`RESERVED_FUTURE`**
Reserved for future definition.

See also:

RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace
This specification defines a Uniform Resource Name namespace for UUIDs, the internal format of UUIDs, and methods of generating UUIDs.

21.20.1. Example

Here are some examples of typical usage of the `uuid` module:

```
>>> import uuid

>>> # make a UUID based on the host ID and current time
>>> uuid.uuid1()
UUID('a8098c1a-f86e-11da-bd1a-00112444be1e')

>>> # make a UUID using an MD5 hash of a namespace UUID and a name
>>> uuid.uuid3(uuid.NAMESPACE_DNS, 'python.org')
UUID('6fa459ea-ee8a-3ca4-894e-db77e160355e')

>>> # make a random UUID
>>> uuid.uuid4()
UUID('16fd2706-8baf-433b-82eb-8c7fada847da')

>>> # make a UUID using a SHA-1 hash of a namespace UUID and a name
>>> uuid.uuid5(uuid.NAMESPACE_DNS, 'python.org')
UUID('886313e1-3b8a-5372-9b90-0c9aee199e5d')
```

```
>>> # make a UUID from a string of hex digits (braces and hyphens ignc
>>> x = uuid.UUID('{00010203-0405-0607-0809-0a0b0c0d0e0f}')

>>> # convert a UUID to a string of hex digits in standard form
>>> str(x)
'00010203-0405-0607-0809-0a0b0c0d0e0f'

>>> # get the raw 16 bytes of the UUID
>>> x.bytes
b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f'

>>> # make a UUID from a 16-byte string
>>> uuid.UUID(bytes=x.bytes)
UUID('00010203-0405-0607-0809-0a0b0c0d0e0f')
```