uuid — Universally Unique Identifiers

Purpose: The uuid module implements Universally Unique Identifiers as described in RFC 4122.

RFC 4122 defines a system for creating universally unique identifiers for resources in a way that does not require a central registrar. UUID values are 128 bits long and, as the reference guide says, “can guarantee uniqueness across space and time.” They are useful for generating identifiers for documents, hosts, application clients, and other situations where a unique value is necessary. The RFC is specifically focused on creating a Uniform Resource Name namespace and covers three main algorithms:

- Using IEEE 802 MAC addresses as a source of uniqueness
- Using pseudo-random numbers
- Using well-known strings combined with cryptographic hashing

In all cases, the seed value is combined with the system clock and a clock sequence value used to maintain uniqueness in case the clock is set backwards.

UUID 1 - IEEE 802 MAC Address

UUID version 1 values are computed using the MAC address of the host. The uuid module uses getnode() to retrieve the MAC value of the current system.

```
# uuid_getnode.py

import uuid

print(hex(uuid.getnode()))
```

If a system has more than one network card, and so more than one MAC, any one of the values may be returned.

```
$ python3 uuid_getnode.py

0xa860b60304d5
```

To generate a UUID for a host, identified by its MAC address, use the uuid1() function. The node identifier argument is optional; leave the field blank to use the value returned by getnode().

```
# uuid_uuid1.py

import uuid

u = uuid.uuid1()

print(u)
print(type(u))
print('bytes   :', repr(u.bytes))
print('hex     :', u.hex)
print('int     :', u.int)
print('urn     :', u.urn)
print('variant :', u.variant)
print('version :', u.version)
print('fields  :', u.fields)
print('  time_low            : ', u.time_low)
print('  time_mid            : ', u.time_mid)
print('  time_hi_version     : ', u.time_hi_version)
print('  clock_seq_hi_variant: ', u.clock_seq_hi_variant)
print('  clock_seq_low       : ', u.clock_seq_low)
print('  node                : ', u.node)
print('  time                : ', u.time)
print('  clock_seq           : ', u.clock_seq)
```

The components of the UUID object returned can be accessed through read-only instance attributes. Some attributes, such as hex, int, and urn, are different representations of the UUID value.

hex, int, and urn, are different representations of the UUID value.

```
$ python3 uuid_uuid1.py

38332b62-2aea-11e8-b103-a860b60304d5
<class 'uuid.UUID'>
bytes   : b'83+b*\xea\x11\xe8\xb1\x03\xa8`\xb6\x03\x04\xd5'
hex     : 38332b622aea11e8b103a860b60304d5
int     : 74702454824994792138317938288475964629
urn     : urn:uuid:38332b62-2aea-11e8-b103-a860b60304d5
variant : specified in RFC 4122
version : 1
fields  : (942877538, 10986, 4584, 177, 3, 185133323977941)
  time_low            :  942877538
  time_mid            :  10986
  time_hi_version     :  4584
  clock_seq_hi_variant:  177
  clock_seq_low       :  3
  node                :  185133323977941
  time                :  137406974088391522
  clock_seq           :  12547
```

Because of the time component, each call to uuid1() returns a new value.

```python
# uuid_uuid1_repeat.py

import uuid

for i in range(3):
    print(uuid.uuid1())
```

In this output, only the time component (at the beginning of the string) changes.

```
$ python3 uuid_uuid1_repeat.py

3842ca28-2aea-11e8-8fec-a860b60304d5
3844cd18-2aea-11e8-aca3-a860b60304d5
3844cdf4-2aea-11e8-ac38-a860b60304d5
```

Because each computer has a different MAC address, running the sample program on different systems will produce entirely different values. This example passes explicit node IDs to simulate running on different hosts.

```python
# uuid_uuid1_othermac.py

import uuid

for node in [0x1ec200d9e0, 0x1e5274040e]:
    print(uuid.uuid1(node), hex(node))
```

In addition to a different time value the node identifier at the end of the UUID also changes.

```
$ python3 uuid_uuid1_othermac.py

3851ea50-2aea-11e8-936d-001ec200d9e0 0x1ec200d9e0
3852caa6-2aea-11e8-a805-001e5274040e 0x1e5274040e
```

UUID 3 and 5 - Name-Based Values

It is also useful in some contexts to create UUID values from names instead of random or time-based values. Versions 3 and 5 of the UUID specification use cryptographic hash values (MD5 or SHA-1, respectively) to combine namespace-specific seed values with names. There are several well-known namespaces, identified by pre-defined UUID values, for working with DNS, URLs, ISO OIDs, and X.500 Distinguished Names. New application-specific namespaces can be defined by generating and saving UUID values.

```python
# uuid_uuid3_uuid5.py

import uuid
```

```
hostnames = ['www.doughellmann.com', 'blog.doughellmann.com']

for name in hostnames:
    print(name)
    print('  MD5   :', uuid.uuid3(uuid.NAMESPACE_DNS, name))
    print('  SHA-1 :', uuid.uuid5(uuid.NAMESPACE_DNS, name))
    print()
```

To create a UUID from a DNS name, pass `uuid.NAMESPACE_DNS` as the namespace argument to `uuid3()` or `uuid5()`:

```
$ python3 uuid_uuid3_uuid5.py

www.doughellmann.com
  MD5   : bcd02e22-68f0-3046-a512-327cca9def8f
  SHA-1 : e3329b12-30b7-57c4-8117-c2cd34a87ce9

blog.doughellmann.com
  MD5   : 9bdabfce-dfd6-37ab-8a3f-7f7293bcf111
  SHA-1 : fa829736-7ef8-5239-9906-b4775a5abacb
```

The UUID value for a given name in a namespace is always the same, no matter when or where it is calculated.

```
# uuid_uuid3_repeat.py

import uuid

namespace_types = sorted(
    n
    for n in dir(uuid)
    if n.startswith('NAMESPACE_')
)
name = 'www.doughellmann.com'

for namespace_type in namespace_types:
    print(namespace_type)
    namespace_uuid = getattr(uuid, namespace_type)
    print(' ', uuid.uuid3(namespace_uuid, name))
    print(' ', uuid.uuid3(namespace_uuid, name))
    print()
```

Values for the same name in the namespaces are different.

```
$ python3 uuid_uuid3_repeat.py

NAMESPACE_DNS
  bcd02e22-68f0-3046-a512-327cca9def8f
  bcd02e22-68f0-3046-a512-327cca9def8f

NAMESPACE_OID
  e7043ac1-4382-3c45-8271-d5c083e41723
  e7043ac1-4382-3c45-8271-d5c083e41723

NAMESPACE_URL
  5d0fdaa9-eafd-365e-b4d7-652500dd1208
  5d0fdaa9-eafd-365e-b4d7-652500dd1208

NAMESPACE_X500
  4a54d6e7-ce68-37fb-b0ba-09acc87cabb7
  4a54d6e7-ce68-37fb-b0ba-09acc87cabb7
```

UUID 4 - Random Values

Sometimes host-based and namespace-based UUID values are not “different enough.” For example, in cases where UUID is intended to be used as a hash key, a more random sequence of values with more differentiation is desirable to avoid collisions in the hash table. Having values with fewer common digits also makes it easier to find them in log files. To add greater differentiation in UUIDs, use `uuid4()` to generate them using random input values.

```
# uuid_uuid4.py
```

```
import uuid

for i in range(3):
    print(uuid.uuid4())
```

The source of randomness depends on which C libraries are available when `uuid` is imported. If `libuuid` (or `uuid.dll`) can be loaded and it contains a function for generating random values, it is used. Otherwise `os.urandom()` or the random module are used.

```
$ python3 uuid_uuid4.py

74695723-65ed-4170-af77-b9f22608535d
db199e25-e292-41cd-b488-80a8f99d163a
196750b3-bbb9-488e-b3ec-62ec0e468bbc
```

Working with UUID Objects

In addition to generating new UUID values, it is possible to parse strings in standard formats to create UUID objects, making it easier to handle comparisons and sorting operations.

```
# uuid_uuid_objects.py

import uuid


def show(msg, l):
    print(msg)
    for v in l:
        print(' ', v)
    print()


input_values = [
    'urn:uuid:f2f84497-b3bf-493a-bba9-7c68e6def80b',
    '{417a5ebb-01f7-4ed5-aeac-3d56cd5037b0}',
    '2115773a-5bf1-11dd-ab48-001ec200d9e0',
]

show('input_values', input_values)

uuids = [uuid.UUID(s) for s in input_values]
show('converted to uuids', uuids)

uuids.sort()
show('sorted', uuids)
```

Surrounding curly braces are removed from the input, as are dashes (`-`). If the string has a prefix containing `urn:` and/or `uuid:`, it is also removed. The remaining text must be a string of 16 hexadecimal digits, which are then interpreted as a UUID value.

```
$ python3 uuid_uuid_objects.py

input_values
  urn:uuid:f2f84497-b3bf-493a-bba9-7c68e6def80b
  {417a5ebb-01f7-4ed5-aeac-3d56cd5037b0}
  2115773a-5bf1-11dd-ab48-001ec200d9e0

converted to uuids
  f2f84497-b3bf-493a-bba9-7c68e6def80b
  417a5ebb-01f7-4ed5-aeac-3d56cd5037b0
  2115773a-5bf1-11dd-ab48-001ec200d9e0

sorted
  2115773a-5bf1-11dd-ab48-001ec200d9e0
  417a5ebb-01f7-4ed5-aeac-3d56cd5037b0
  f2f84497-b3bf-493a-bba9-7c68e6def80b
```

See also

- Standard library documentation for uuid
- Python 2 to 3 porting notes for uuid
- **RFC 4122** – A Universally Unique Identifier (UUID) URN Namespace

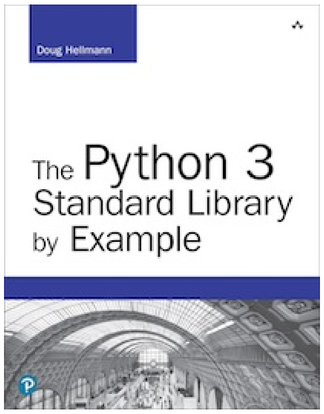
webbrowser — Displays web pages　　json — JavaScript Object Notation

Quick Links

This page was last updated 2018-03-18.

Navigation

webbrowser — Displays web pages
json — JavaScript Object Notation



Get the book

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

- Module Index
- Index

Other Writing

- Blog
- The Python Standard Library By Example