NAME

uuid — generate, convert, and decode Universally Unique IDentifiers

SYNOPSIS

uuid [**-n** *count*] [**-o** *outfile*] [**-r**|**-F BIN**|**STR**|**SIV**] [**-v 1**] [**-m1**]
uuid [**-n** *count*] [**-o** *outfile*] [**-r**|**-F BIN**|**STR**|**SIV**] **-v 3**
uuid|**nil**|**max**|**ns:**{**DNS**|**URL**|**OID**|**X500**} *data*
uuid [**-n** *count*] [**-o** *outfile*] [**-r**|**-F BIN**|**STR**|**SIV**] **-v 4**
uuid [**-n** *count*] [**-o** *outfile*] [**-r**|**-F BIN**|**STR**|**SIV**] **-v 5**
uuid|**nil**|**max**|**ns:**{**DNS**|**URL**|**OID**|**X500**} *data*
uuid [**-n** *count*] [**-o** *outfile*] [**-r**|**-F BIN**|**STR**|**SIV**] **-v 6**
uuid [**-n** *count*] [**-o** *outfile*] [**-r**|**-F BIN**|**STR**|**SIV**] **-v 7**

uuid -d [**-o** *outfile*] [**-F** **STR**|**SIV**] *uuid*
uuid -d [**-o** *outfile*] [**-r**|**-F BIN**|**STR**|**SIV**] **-**

DESCRIPTION

By default, generates a UUID (Universally Unique Identifier), of a version specified by **-v** (default **1**) in the canonical **STR**ing format.

With **-d**, decodes *uuid* (or reads it in from the standard input stream if "**-**"), yielding something akin to

```
  encode: STR:     92a3d3de-6bbf-11ef-9f5b-774ebb537938
          SIV:     194917928982963228599463962120788605240
  decode: variant: DCE 1.1, ISO/IEC 11578:1996
          version: 1 (time and node based)
          content: time:  2024-09-05 19:46:41.491043.0 UTC
                   clock: 8027 (usually random)
                   node:  77:4e:bb:53:79:38 (local multicast)
```

or

```
  encode: STR:     7e017d98-cecc-41f6-8030-552d23cd38e6
          SIV:     167490467173639937831846401373276813542
  decode: variant: DCE 1.1, ISO/IEC 11578:1996
          version: 4 (random data based)
          content: 7E:01:7D:98:CE:CC:01:F6:00:30:55:2D:23:CD:38:E6
                   (no semantics: random data only)
```

A UUID is a 128-bit (16-byte) number whose format (method of generation) makes it very likely that it will be unique, both spatially and temporally. This makes them well-suited for identifying anything from ephemera like database queries to databases themselves.

The three principal **-F**ormats are:

STRing	`01234567-890a-bcde-f012-3456789abcde`	(hexadecimal string broken up by dashes)
SIV (Single Integer Value)	128-bit decimal integer	
BINary or "raw"	the 16 bytes of the UUID in big endian order (most significant first)	

UUIDs come in a few **-v**ersions:

1 based on the current time (to the precision of 100ns), and the host's MAC address (this implementation uses the MAC address of the first NIC it finds, unless **-m**, in which case random data is substituted for the MAC)

6 very similar, but the most significant (slowest-changing) parts of the timestamp are stored first, which improves locality in some database applications

7 contains the current *UNIX* time — versions **1** and **6** encode a UUID time which is off by like 400 years — (to the precision of 1ms), then 10 bytes of random data.
This version should be preferred to 1 and 6.

3 hashes the "namespace" (first argument, either a **STR**ing-format UUID or a well-known name (see below)) and the "name" (second argument, arbitrary data) with the `MD5`(3) digest, and uses that directly (a few branding bytes are replaced to identify the format)

5 the same as version **3**, but uses the first 16 bytes of the SHA1(3) digest

4 just random data

The namespaces understood when generating version 3 and 5 UUIDs are:

uuid	(any valid **STR**ingified UUID)
nil	00000000-0000-0000-0000-000000000000 — the special all-zero UUID
max	ffffffff-ffff-ffff-ffff-ffffffffffff — the special all-bits-set/sorts-after-everything sentinel UUID
ns:DNS	6ba7b810-9dad-11d1-80b4-00c04fd430c8 — for fully-qualified domain names
ns:URL	6ba7b811-9dad-11d1-80b4-00c04fd430c8 — for URLs
ns:OID	6ba7b812-9dad-11d1-80b4-00c04fd430c8 — for ISO OIDs
ns:X500	6ba7b814-9dad-11d1-80b4-00c04fd430c8 — for X.500 Distinguished Names

OPTIONS

-n *count* — Generate *count* UUIDs. Defaults to **1**.

-o *outfile* — Write to *outfile*.

-F BIN|STR|SIV — Produce output in the given format. Defaults to **STR**.

-r — **-F BIN**

-v 1|3|4|5|6|7 — Version to generate. Defaults to **1**.

-m — **-v 1|6** only: ignore the current host's MAC addresses, use random data instead (this may still happen if the if a MAC can't be determined or all MACs are multicast).

-1 — **-v 1** only, *count* >1 only: generate each UUID independently. Version 1 UUIDs have a field that increases monotonically within a session; thus, for example, in

```
$ uuid -n 4
366ab5a3-6bc4-11ef-a31a-0026b986fdd4
366ab5ce-6bc4-11ef-a31b-0026b986fdd4
366ab5f8-6bc4-11ef-a31c-0026b986fdd4
366ab621-6bc4-11ef-a31d-0026b986fdd4
                      ^^^^
```

the highlighted column *starts* random, but then increments. **-1** generates each UUID de novo.

-d — Decode and parse *uuid*. **-F** sets which format to *read*. **-F BIN** is only available when reading from the standard input stream ("**-**").

EXAMPLES

A web site can be uniquely identified with a version 4 (SHA-1), namespace **ns:DNS** UUID, and a web-page — **ns:URL** (note that this information cannot be extracted, and only serves as a way to avoid collisions for otherwise-identical names):

```
$ uuid -v 4 ns:DNS hinfo.network.
f50b485f-ac66-591d-b95f-2c946c5a5668
$ uuid -v 4 ns:URL https://hinfo.network
8cebc56f-51d0-5323-a031-1b9258de14f8
$ uuid -d 8cebc56f-51d0-5323-a031-1b9258de14f8
encode: STR:     f50b485f-ac66-591d-b95f-2c946c5a5668
        SIV:     325719442146270326702531202385345926760
decode: variant: DCE 1.1, ISO/IEC 11578:1996
        version: 5 (name based, SHA-1)
        content: F5:0B:48:5F:AC:66:09:1D:39:5F:2C:94:6C:5A:56:68
                 (not decipherable: truncated SHA-1 message digest only)

$ uuid -v 7
8cebc56f-51d0-5323-a031-1b9258de14f8
$ uuid -d 8cebc56f-51d0-5323-a031-1b9258de14f8
encode: STR:     0191c41a-9de1-7f96-b8cb-88e4348ee74c
        SIV:     208608850134876434955189021385312852 4
decode: variant: DCE 1.1, ISO/IEC 11578:1996
        version: 7 (UNIX time + random data)
```

```
content: time:   2024-09-05 21:32:44.385 UTC
         random: 0F:96:38:CB:88:E4:34:8E:E7:4C
```

SEE ALSO

uuid(3), OSSP::uuid(3)

STANDARDS

RFC 9562: Universally Unique IDentifiers (UUIDs): https://datatracker.ietf.org/doc/html/rfc9562 supersedes all previous standards with an unbecoming brevity.

There are many Versions, but only one useful Variant.

NAME

uuid-config — OSSP uuid API build utility

SYNOPSIS

uuid [**--help**] [**--version**] [**--prefix**] [**--exec-prefix**] [**--bindir**] [**--libdir**] [**--includedir**] [**--mandir**] [**--datadir**] [**--acdir**] [**--cflags**] [**--ldflags**] [**--libs**]

DESCRIPTION

The **uuid** program is a little helper utility for easy configuring and building of applications based on the uuid(3) library. It can be used to query the C compiler and linker flags which are required to correctly compile and link the application against the uuid(3) library.

OSSP uuid also ships pkg-config(1) definition (as uuid.pc). Use that instead.

OPTIONS

--help	Prints the short usage information.
--version	Prints the version number and date of the installed uuid(3) library.
--prefix	Prints the installation prefix of architecture independent files
--exec-prefix	Prints the installation prefix of architecture dependent files.
--bindir	Prints the installation directory of binaries.
--libdir	Prints the installation directory of libraries.
--includedir	Prints the installation directory of include headers.
--mandir	Prints the installation directory of manual pages.
--datadir	Prints the installation directory of shared data.
--acdir	Prints the installation directory of **autoconf** data.
--cflags	Prints the C compiler flags which are needed to compile the uuid(3)-based application. The output is usually added to the CFLAGS uuidiable of the applications Makefile.
-ldflags	Prints the linker flags (**-L**) which are needed to link the application with the uuid(3) library. The output is usually added to the LDFLAGS uuidiable of the applications Makefile.
-libs	Prints the library flags (**-l**) which are needed to link the application with the C uuid(3) library. The output is usually added to the LIBS uuidiable of the applications Makefile.

EXAMPLES

```
CC      = cc
CFLAGS  = -O `uuid-config --cflags`
LDFLAGS = `uuid-config --ldflags`
LIBS    = -lm `uuid-config --libs`

all: foo
foo: foo.o
       $(CC) $(LDFLAGS) -o foo foo.o $(LIBS)
foo.o: foo.c
       $(CC) $(CFLAGS) -c foo.c
```

SEE ALSO

uuid(1), uuid(3), OSSP::uuid(3)

NAME

uuid++ — deprecated unusable API to the C++ OSSP Universally Unique Identifier library

SYNOPSIS

```
#include <uuid++.hh>

class [[deprecated("the OSSP uuid++ library is broken, unsalvageable, and strictly wors
public:
  uuid();
  uuid(const uuid   &obj);
  uuid(const uuid_t *obj); // copy from C library uuid_t
  uuid(const void   *bin); // import from UUID_FMT_BIN
  uuid(const char   *str); // import from UUID_LEN_STR (or UUID_LEN_SIV if that fails)
  ~uuid();

  uuid& operator=(const uuid   &obj);
  uuid& operator=(const uuid_t *obj); // copy from C library uuid_t
  uuid& operator=(const void   *bin); // import from UUID_FMT_BIN
  uuid& operator=(const char   *str); // import from UUID_LEN_STR (or UUID_LEN_SIV if th
  uuid clone();                       // crash (or produce a broken UUID if it doesn't)

  void load(const char *name);        // replace with uuid_load()
  void make(unsigned int mode, ...);  // replace with uuid_make()

  int isnil();                        // replace with uuid_isnil()
  int compare(const uuid &obj);       // replace with uuid_compare()
  // == != < <= > >=  operators

  void  import(const void *bin);     // replace with uuid_import(UUID_FMT_BIN)
  void  import(const char *str);     // replace with uuid_import(UUID_FMT_STR), uuid_imp

  void* binary();                    // replace with uuid_export(UUID_FMT_BIN); always r
  char* string();                    // replace with uuid_export(UUID_FMT_STR); always r
  char* integer();                   // replace with uuid_export(UUID_FMT_SIV); always r
  char* summary();                   // replace with uuid_export(UUID_FMT_TXT); always r

  unsigned long version();           // replace with uuid_version()
};

class [[deprecated("the OSSP uuid++ library is broken, unsalvageable, and strictly wors
public:
  uuid_error_t() : uuid_error_t(UUID_RC_OK) {}
  uuid_error_t(uuid_rc_t code);
  ~uuid_error_t();

  void      code(uuid_rc_t code); // setter
  uuid_rc_t code();               // getter

  char *    string();             // replace with uuid_error()
};
```

DESCRIPTION

Replace with uuid(3).

NAME

uuid — Universally Unique IDentifier

SYNOPSIS

```
#include <uuid.h>
cc $(pkg-config --libs --cflags uuid) ...
```

DESCRIPTION

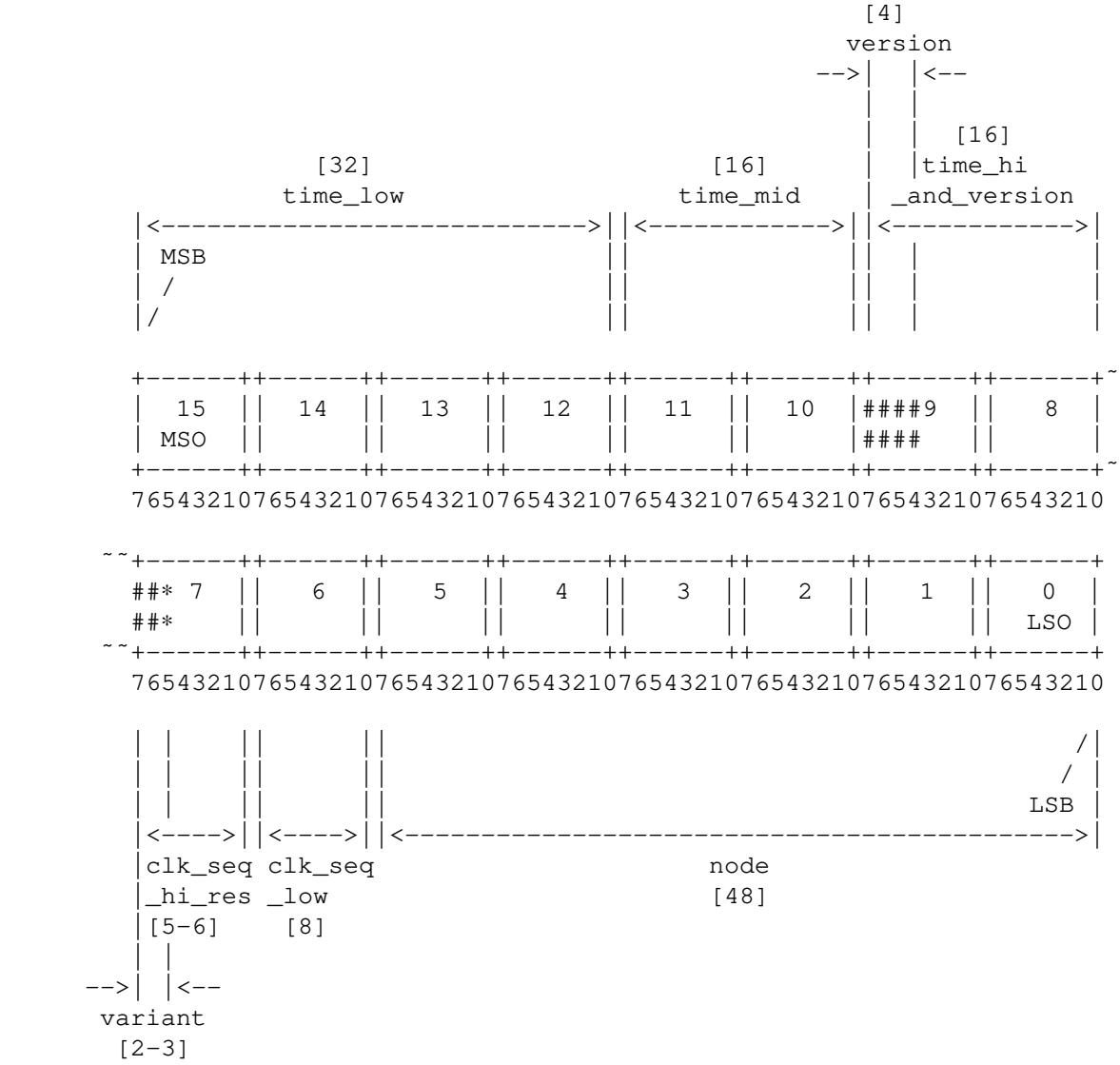
OSSP **uuid** is a ISO-C:1999 application programming interface (API) and corresponding command line interface (CLI) for the generation of RFC 9562, ISO/IEC 11578:1996 and IETF RFC-4122 compliant *Universally Unique Identifier* (UUID). It supports DCE 1.1 variant UUIDs of version 1 (time and node based), version 3 (name based, MD5), version 4 (random number based) and version 5 (name based, SHA-1). Additional API bindings are provided for the Perl:5 language. Optional backward compatibility exists for the ISO-C DCE-1.1 and Perl Data::UUID APIs.

UUIDs are 128 bit numbers which are intended to have a high likelihood of uniqueness over space and time and are computationally difficult to guess. They are globally unique identifiers which can be locally generated without contacting a global registration authority. UUIDs are intended as unique identifiers for both mass tagging objects with an extremely short lifetime and to reliably identifying very persistent objects across a network.

This is the ISO-C application programming interface (API) of OSSP **uuid**.

UUID Binary Representation

According to RFC 9562, ISO/IEC 11578:1996 and IETF RFC-4122 standards, a DCE 1.1 variant UUID is a 128 bit number defined out of 7 fields, each field a multiple of an octet in size and stored in network byte order:



An example of a UUID binary representation is the octet stream `0xF8 0x1D 0x4F 0xAE 0x7D 0xEC 0x11 0xD0 0xA7 0x65 0x00 0xA0 0xC9 0x1E 0x6B 0xF6`. The binary representation format is exactly what the OSSP **uuid** API functions **uuid_import**() and **uuid_export**() deal with under `UUID_FMT_BIN`.

UUID ASCII String Representation

According to RFC 9562, ISO/IEC 11578:1996 and IETF RFC-4122 standards, a DCE 1.1 variant UUID is represented as an ASCII string consisting of 8 hexadecimal digits followed by a hyphen, then three groups of 4 hexadecimal digits each followed by a hyphen, then 12 hexadecimal digits. Formally, the string representation is defined by the following grammar:

```
uuid                          = <time_low> "-"
                                <time_mid> "-"
                                <time_high_and_version> "-"
                                <clock_seq_high_and_reserved>
                                <clock_seq_low> "-"
                                <node>
time_low                      = 4*<hex_octet>
time_mid                      = 2*<hex_octet>
time_high_and_version         = 2*<hex_octet>
clock_seq_high_and_reserved = <hex_octet>
clock_seq_low                 = <hex_octet>
node                          = 6*<hex_octet>
hex_octet                     = <hex_digit> <hex_digit>
hex_digit                     = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
                               |"a"|"b"|"c"|"d"|"e"|"f"
                               |"A"|"B"|"C"|"D"|"E"|"F"
```

An example of a UUID string representation is the ASCII string "`f81d4fae-7dec-11d0-a765-00a0c91e6bf6`". The string representation format is exactly what the OSSP **uuid** API functions **uuid_import**() and **uuid_export**() deal with under `UUID_FMT_STR`.

Notice: a corresponding URL can be generated out of a ASCII string representation of an UUID by prefixing with **urn:uuid:** as in "`urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6`".

UUID Single Integer Value Representation

According to the ISO/IEC 11578:1996 and ITU-T Rec. X.667 standards, a DCE 1.1 variant UUID can be also represented as a single integer value consisting of a decimal number with up to 39 digits.

An example of a UUID single integer value representation is the decimal number "`329800735698586629295641978511506172918`". The string representation format is exactly what the OSSP **uuid** API functions **uuid_import**() and **uuid_export**() deal with under `UUID_FMT_SIV`.

Notice: a corresponding ISO OID can be generated under the "{joint-iso-itu-t(2) uuid(25)}" arc out of a single integer value representation of a UUID by prefixing with **2.25.**. An example OID is "`2.25.329800735698586629295641978511506172918`". Additionally, an URL can be generated by further prefixing with **urn:oid:** as in "`urn:oid:2.25.329800735698586629295641978511506172918`".

UUID Variants and Versions

A UUID has a variant and version. The variant defines the layout of the UUID. The version defines the content of the UUID. The UUID variant supported in OSSP **uuid** is DCE 1.1 variant only. DCE 1.1 UUID variant versions supported in OSSP **uuid** are:

Version **1** (time and node based)

These are the classical UUIDs, created out of a 60-bit system time, a 14-bit local clock sequence and 48-bit system MAC address. The MAC address can be either the real one of a physical network interface card (NIC) or a random multi-cast MAC address. Version 1 UUIDs are usually used as one-time global unique identifiers.

Version **3** (name based, MD5)
: These are UUIDs which are based on the 128-bit MD5 message digest of the concatenation of a 128-bit namespace UUID and a name string of arbitrary length. Version 3 UUIDs are usually used for non-unique but repeatable message digest identifiers.

Version **4** (random data based)
: These are UUIDs which are based on just 128-bit of random data. Version 4 UUIDs are usually used as one-time local unique identifiers.

Version **5** (name based, SHA-1)
: These are UUIDs which are based on the 160-bit SHA-1 message digest of the concatenation of a 128-bit namespace UUID and a name string of arbitrary length. Version 5 UUIDs are usually used for non-unique but repeatable message digest identifiers.

Version **5** (name based, SHA-1)
: These are UUIDs which are based on the 160-bit SHA-1 message digest of the concatenation of a 128-bit namespace UUID and a name string of arbitrary length. Version 5 UUIDs are usually used for non-unique but repeatable message digest identifiers.

Version **6** (time (reverse) and node based)
: These are compatible with and very similar to Version **1** UUIDs, except the time is stored from the most significant parts to the least significant parts. This improves locality in some database implementations.

Version **7** (UNIX time based)
: These UUIDs consist of a 48-bit UNIX time (with 1ms precision) in big-endian, then 10 bytes of random data. RFC 9562 *recommends* these for new designs.

UUID Uniqueness
: Version **1/6** UUIDs are guaranteed to be unique through combinations of hardware addresses, time stamps and random seeds. There is a reference in the UUID to the hardware (MAC) address of the first network interface card (NIC) on the host which generated the UUID — this reference is intended to ensure the UUID will be unique in space as the MAC address of every network card is assigned by a single global authority (IEEE) and is guaranteed to be unique. The next component in a UUID is a timestamp which, as clock always (should) move forward, will be unique in time. Just in case some part of the above goes wrong (the hardware address cannot be determined or the clock moved steps backward), there is a random clock sequence component placed into the UUID as a "catch-all" for uniqueness.

  Version **3/5** UUIDs are guaranteed to be inherently globally unique if the combination of namespace and name used to generate them is unique.

  Version **4** UUIDs are not guaranteed to be globally unique, because they are generated out of locally gathered random numbers only. Nevertheless there is still a high likelihood of uniqueness over space and time and that they are computationally difficult to guess.

  Version **7** UUIDs attempt to guarantee uniqueness by subdividing each millisecond of real time into $2^{80}(10^{24})$ possible values.

Nil UUID
: There is a special *Nil* UUID consisting of all octets set to zero in the binary representation. It can be used as a special UUID value which does not conflict with real UUIDs.

Max UUID
: Similarly to the *Nil* UUID, the *Max* UUID consists of all octets set to 0xFF (255, all bits set) in the binary representation. It can be used as a special UUID value which does not conflict with real UUIDs, but sorts after everything else.

APPLICATION PROGRAMMING INTERFACE

The ISO-C Application Programming Interface (API) of OSSP **uuid** consists of the following components.

CONSTANTS

The following constants are provided:

`UUID_VERSION`
: The hexadecimal encoded OSSP **uuid** version. This allows compile-time checking of the OSSP **uuid** version. For run-time checking use **uuid_version**() instead.

  The current value `0x106203` encodes `1.6.3`.

`UUID_LEN_BIN`, `UUID_LEN_STR`, `UUID_LEN_SIV`
: The number of octets of the UUID binary and string representations. Notice that the lengths of the string representation (`UUID_LEN_STR`) and the lengths of the single integer value representation (`UUID_LEN_SIV`) does *not* include the necessary NUL termination character.

`UUID_MAKE_V1`, `UUID_MAKE_V3`, `UUID_MAKE_V4`, `UUID_MAKE_V5`, `UUID_MAKE_V6`, `UUID_MAKE_V7`, `UUID_MAKE_MC`
: The *mode* bits for use with **uuid_make**(). `UUID_MAKE_V`*n* specify which UUID version to generate. `UUID_MAKE_MC` forces the use of a random multi-cast MAC address instead of the real physical MAC address in version 1 and 6 UUIDs.

`UUID_RC_OK`, `UUID_RC_ARG`, `UUID_RC_MEM`, `UUID_RC_SYS`, `UUID_RC_INT`, `UUID_RC_IMP`
: The possible numerical return-codes of API functions. The `UUID_RC_OK` indicates success, the others indicate errors. Use **uuid_error**() to translate them into string versions.

`UUID_FMT_BIN`, `UID_FMT_STR`, `UID_FMT_SIV`, `UID_FMT_TXT`
: The *fmt* formats for use with **uuid_import**() and **uuid_export**():
  - `UUID_FMT_BIN` indicates the UUID binary representation (of length `UUID_LEN_BIN`),
  - `UUID_FMT_STR` indicates the UUID string representation (of length `UUID_LEN_STR`),
  - `UUID_FMT_SIV` indicates the UUID single integer value representation (of maximum length `UUID_LEN_SIV`), and
  - `UUID_FMT_TXT` indicates the textual description (of arbitrary length) of a UUID (this powers `uuid`(1) **-d**).

FUNCTIONS

uuid_rc_t **uuid_create**(*uuid_t **uuid*)
: Create a new UUID object and store a pointer to it in **uuid*. A UUID object consists of an internal representation of a UUID, potential internal RNG context, and timestamp information. The initial UUID is the *Nil* UUID.

uuid_rc_t **uuid_destroy**(*uuid_t *uuid*)
: Destroy UUID object *uuid*.

uuid_rc_t **uuid_clone**(*const uuid_t *uuid*, *uuid_t **uuid_clone*)
: Clone UUID object *uuid* and store new UUID object in **uuid_clone*.

uuid_rc_t **uuid_isnil**(*const uuid_t *uuid*, *int *result*)
: Checks whether the UUID in uuid is the *Nil* UUID. If this is the case, it returns **1** in **result*. Else it returns **false** in **result*.

uuid_rc_t **uuid_ismax**(*const uuid_t *uuid*, *int *result*)
: Checks whether the UUID in uuid is the *Max* UUID. If this is the case, it returns **1** in **result*. Else it returns **false** in **result*.

uuid_rc_t **uuid_compare**(*const uuid_t *uuid*, *const uuid_t *uuid2*, *int *result*)
: Compares the order of the two UUIDs and returns the result in **result*: **-1** if *uuid* < *uuid2*, **0** if *uuid* = *uuid2*, and **+1** if *uuid* > *uuid2*.

uuid_rc_t **uuid_import**(*uuid_t *uuid*, *uuid_fmt_t fmt*, *const void *data_ptr*, *size_t data_len*)
: Imports *uuid* from an external representation of format *fmt*. The data is read from the buffer at *data_ptr* which contains at least *data_len* bytes.

  The format of the external representation is specified by *fmt* and the minimum expected length in *data_len* depends on it. Valid values for *fmt* are `UUID_FMT_BIN`, `UUID_FMT_STR`, and `UUID_FMT_SIV`.

uuid_rc_t **uuid_export**(*const uuid_t *uuid*, *uuid_fmt_t fmt*, *void *data_ptr*, *size_t *data_len*)
Exports *uuid* into an external representation of format *fmt*. Valid values for *fmt* are UUID_FMT_BIN, UUID_FMT_STR, UUID_FMT_SIV, and UUID_FMT_TXT.

The data is written to the buffer located at **(type **)data_ptr*, where *type* is *uint8_t* for UUID_FMT_BIN and *char* for other formats.

The buffer has to have room for at least **data_len* bytes. If **(type **)data_ptr* is NULL, *data_len* is ignored as input and a new buffer is allocated with malloc(3) and returned in **(type **)data_ptr*.

If *data_len* is not NULL, the number of available bytes in the buffer has to be provided in **data_len*. The number of actually written bytes are returned in **data_len* again. The minimum required buffer length depends on the external representation as specified by *fmt* (see above). For UUID_FMT_TXT, a buffer of unspecified length is required, and hence it is recommended to allow OSSP **uuid** to allocate the buffer as necessary.

uuid_rc_t **uuid_load**(*uuid_t *uuid*, *const char *name*)
Loads a pre-defined UUID value into *uuid*. The following *name* arguments are currently known:

nil	00000000-0000-0000-0000-000000000000
max	ffffffff-ffff-ffff-ffff-ffffffffffff
ns:DNS	6ba7b810-9dad-11d1-80b4-00c04fd430c8
ns:URL	6ba7b811-9dad-11d1-80b4-00c04fd430c8
ns:OID	6ba7b812-9dad-11d1-80b4-00c04fd430c8
ns:X500	6ba7b814-9dad-11d1-80b4-00c04fd430c8

ns:*XXX* are names of pre-defined name-space UUIDs for use in the generation of DCE 1.1 version 3 and version 5 UUIDs.

uuid_rc_t **uuid_make**(*uuid_t *uuid*, *unsigned int mode*, ...)
Generates a new UUID in *uuid* according to *mode* and optional arguments (dependent on *mode*).

If *mode* contains the UUID_MAKE_V1 or UUID_MAKE_V6 bit, a DCE 1.1 variant UUID of version **1** or **6** is generated. Then optionally the bit UUID_MAKE_MC forces the use of random multi-cast MAC address instead of the real physical MAC address (the default). The UUID is generated out of the 60-bit current system time, a 12-bit clock sequence (initialised to random, then incremental in v**1**; always random in v**6**) and the 48-bit MAC address.

If *mode* contains the UUID_MAKE_V3 or UUID_MAKE_V5 bit, a DCE 1.1 variant UUID of version **3** or **5** is generated and two additional arguments are expected: first, a namespace UUID object (*uuid_t **). Second, a name string of arbitrary length (*const char **). The UUID is generated out of the 128-bit MD5 or 160-bit SHA-1 from the concatenated octet stream of namespace UUID and name string.

If *mode* contains the UUID_MAKE_V4 bit, a DCE 1.1 variant UUID of version 4 is generated. The UUID is generated out of 128-bit random data.

If *mode* contains the UUID_MAKE_V7 bit, a DCE 1.1 variant UUID of version 7 is generated. The UUID is generated out of 48 bits of time and 80 bits of random data.

*char ** **uuid_error**(*uuid_rc_t rc*)
Returns a constant string representation corresponding to the return-code *rc* for use in displaying OSSP **uuid** errors.

unsigned long **uuid_version**()
Returns the hexadecimal encoded OSSP **uuid** version as compiled into the library object files. This allows run-time checking of the OSSP **version**. For compile-time checking use UUID_VERSION instead.

EXAMPLES

The following shows an example usage of the API. Error handling is omitted for code simplification and has to be re-added for production code.

```
/* generate a RFC 9562 v1 UUID from system environment */
char *uuid_v1() {
        uuid_t *uuid;
        uuid_create(&uuid);
        uuid_make(uuid, UUID_MAKE_V1);

        char *str = NULL;
        uuid_export(uuid, UUID_FMT_STR, &str, NULL);
        uuid_destroy(uuid);
        return str;
}

/* generate a RFC 9562 v3 UUID from an URL */
char *uuid_v3(const char *url) {
        uuid_t *uuid_ns;
        uuid_create(&uuid_ns);
        uuid_load(uuid_ns, "ns:URL");

        uuid_t *uuid;
        uuid_create(&uuid);
        uuid_make(uuid, UUID_MAKE_V3, uuid_ns, url);

        char *str = NULL;
        uuid_export(uuid, UUID_FMT_STR, &str, NULL);
        uuid_destroy(uuid_ns);
        uuid_destroy(uuid);
        return str;
}
```

SEE ALSO

uuid(1), OSSP::uuid(3)

The following are references to UUID documentation and specifications:

K. Davis, B. Peabody, and P. Leach, *Universally Unique IDentifiers (UUIDs)*, RFC-9562 ISSN 2070-1721, https://datatracker.ietf.org/doc/html/rfc9562, IETF Cisco Systems Uncloud University of Washington. This is the current UUID standard.

P. Leach, M. Mealling, and R. Salz, *A Universally Unique IDentifier (UUID) URN Namespace*, RFC-4122, http://www.ietf.org/rfc/rfc4122.txt, IETF, July 2005.

Information Technology – Open Systems Interconnection (OSI), Procedures for the operation of OSI Registration Authorities: Generation and Registration of Universally Unique Identifiers (UUIDs) and their Use as ASN.1 Object Identifier Components, 9834-8:2004 / ITU-T Rec. X.667 2004, http://www.itu.int/ITU-T/studygroups/com17/oid/X.667-E.pdf, ISO/IEC, December 2004.

"Universally Unique Identifier", *DCE 1.1: Remote Procedure Call*, Document Number C706, http://www.opengroup.org/publications/catalog/c706.htm, Open Group Technical Standard, August 1997.

"6.4.1 Node Field Generation Without the IEEE 802 Address", *HTTP Extensions for Distributed Authoring (WebDAV)*, RFC-2518, http://www.ietf.org/rfc/rfc2518.txt, IETF, February 1999.

"DCE 1.1 compliant UUID functions", *FreeBSD manual pages uuid(3) and uuidgen(2)*, http://www.freebsd.org/cgi/man.cgi?query=uuid&manpath=FreeBSD+6.0-RELEASE.

HISTORY

OSSP **uuid** was implemented in January 2004 by Ralf S. Engelschall <rse@engelschall.com>. It was prompted by the use of UUIDs in the OSSP **as** and **OpenPKG** projects. It is a clean room implementation intended to be strictly standards compliant and maximally portable.

NAME

Data::UUID — OSSP uuid Backward Compatibility Perl Binding

DESCRIPTION

uuid is the OSSP **uuid** backward compatibility Perl binding to the API of the original **uuid** module. It allows other **uuid** based Perl modules to run with **OSSP::uuid** without changes.

SEE ALSO

OSSP::uuid(3perl)

HISTORY

The backward compatibility Perl binding **uuid** to OSSP **uuid** was originally implemented in 2004 by Piotr Roszatycki <dexter@debian.org>. It was later cleaned up and speed optimized in December 2005 by David Wheeler <david@justatheory.com>.

NAME

OSSP::uuid — OSSP uuid Perl Binding

DESCRIPTION

OSSP **uuid** is a ISO-C:1999 application programming interface (API) and corresponding command line interface (CLI) for the generation of RFC 9562, ISO/IEC 11578:1996 and IETF RFC-4122 compliant *Universally Unique Identifier* (UUID). It supports DCE 1.1 variant UUIDs of version 1 (time and node based), version 3 (name based, MD5), version 4 (random number based) and version 5 (name based, SHA-1). Additional API bindings are provided for the Perl:5 language. Optional backward compatibility exists for the ISO-C DCE-1.1 and Perl Data::UUID APIs.

uuid is the Perl binding to the OSSP **uuid** API. Three variants are provided:

TIE-STYLE API

The TIE-style API is a functionality-reduced wrapper around the OO-style API and intended for very high-level convenience programming:

```
use OSSP::uuid;
tie my $uuid, 'OSSP::uuid::tie', $mode, ...;
$uuid = [ $mode, ... ];
print "UUID=$uuid\n";
untie $uuid;
```

OO-STYLE API

The OO-style API is a wrapper around the C-style API and intended for high-level regular programming.

```
use OSSP::uuid;
my $uuid = new OSSP::uuid;
$uuid->load ($name);
$uuid->make ($mode, ...);
$result = $uuid->isnil();
$result = $uuid->compare($uuid2);
$uuid->import($fmt, $data_ptr);
$data_ptr = $uuid->export($fmt);
[()$str[, $rc)] = $uuid->error();
$ver = $uuid->version();
undef $uuid;
```

Additionally, the strings "v1", "v3", "v4", "v5", "v6", "v7", and "mc" can be used in $mode and the strings "bin", "str", and "txt" $fmt.

C-STYLE API

The C-style API is a direct mapping of the OSSP **uuid** ISO-C API to Perl and is intended for low-level programming. See uuid(3) for a description of the functions and their expected arguments.

```
use OSSP::uuid qw(:all);
my $uuid; $rc = uuid_create($uuid);
$rc = uuid_load($uuid, $name);
$rc = uuid_make($uuid, $mode, ...);
$rc = uuid_isnil($uuid, $result);
$rc = uuid_ismax($uuid, $result);
$rc = uuid_compare($uuid, $uuid2, $result);
$rc = uuid_import($uuid, $fmt, $data_ptr, $data_len);
$rc = uuid_export($uuid, $fmt, $data_ptr, $data_len);
$str = uuid_error($rc);
$ver = uuid_version();
$rc = uuid_destroy($uuid);
```

Additionally, the following constants are exported for use in $rc, $mode, $fmt, and $ver: UUID_VERSION, UUID_LEN_BIN, UUID_LEN_STR, UUID_RC_OK, UUID_RC_ARG, UUID_RC_MEM, UUID_RC_SYS, UUID_RC_INT, UUID_RC_IMP, UUID_MAKE_V1, UUID_MAKE_V3, UUID_MAKE_V4, UUID_MAKE_V5, UUID_MAKE_MC, UUID_MAKE_V6, UUID_MAKE_V7, UUID_FMT_BIN, UUID_FMT_STR, UUID_FMT_SIV, UUID_FMT_TXT.

EXAMPLES

The following two examples create the version 3 UUID `02d9e6d5-9467-382e-8f9b-9300a64ac3cd`, both via the OO-style and the C-style API. Error handling is omitted here for easier reading, but has to be added for production-quality code.

```
#   TIE-style API (very high-level)
use OSSP::uuid;
tie my $uuid, 'OSSP::uuid::tie';
$uuid = [ "v1" ];
print "UUIDs: $uuid, $uuid, $uuid\n";
$uuid = [ "v3", "ns:URL", "http://www.ossp.org/" ];
print "UUIDs: $uuid, $uuid, $uuid\n";
untie $uuid;

#   OO-style API (high-level)
use OSSP::uuid;
my $uuid    = new OSSP::uuid;
my $uuid_ns = new OSSP::uuid;
$uuid_ns->load("ns:URL");
$uuid->make("v3", $uuid_ns, "http://www.ossp.org/");
undef $uuid_ns;
my $str = $uuid->export("str");
undef $uuid;
print "$str\n";

#   C-style API (low-level)
use OSSP::uuid qw(:all);
my $uuid;    uuid_create($uuid);
my $uuid_ns; uuid_create($uuid_ns);
uuid_load($uuid_ns, "ns:URL");
uuid_make($uuid, UUID_MAKE_V3, $uuid_ns, "http://www.ossp.org/");
uuid_destroy($uuid_ns);
my $str; uuid_export($uuid, UUID_FMT_STR, $str, undef);
uuid_destroy($uuid);
print "$str\n";
```

SEE ALSO

uuid(1), uuid-config(1), uuid(3)

HISTORY

The Perl binding **uuid** to OSSP **uuid** was implemented in November 2004 by Ralf S. Engelschall <rse@engelschall.com>.