# uuid [![Build Status](https://secure.travis-ci.org/kelektiv/node-uuid.svg?branch=master)](http://travis-ci.org/kelektiv/node-uuid) # Simple, fast generation of [RFC4122](http://www.ietf.org/rfc/rfc4122.txt) UUIDS. Features: \* Support for version 1, 3, 4 and 5 UUIDs \* Cross-platform \* Uses cryptographically-strong random number APIs (when available) \* Zero-dependency, small footprint (... but not [this small](https://gist.github.com/982883)) [\*\*Deprecation warning\*\*: The use of `require('uuid')` is deprecated and will not be supported after version 3.x of this module. Instead, use `require('uuid/[v1|v3|v4|v5]')` as shown in the examples below.] ## Quickstart - CommonJS (Recommended) ```shell npm install uuid ``` Then generate your uuid version of choice ... Version 1 (timestamp): ```javascript const uuidv1 = require('uuid/v1'); uuidv1(); // ⇨ '2c5ea4c0-4067-11e9-8bad-9b1deb4d3b7d' ``` Version 3 (namespace): ```javascript const uuidv3 = require('uuid/v3'); // ... using predefined DNS namespace (for domain names) uuidv3('hello.example.com', uuidv3.DNS); // ⇨ '9125a8dc-52ee-365b-a5aa-81b0b3681cf6' // ... using predefined URL namespace (for, well, URLs) uuidv3('http://example.com/hello', uuidv3.URL); // ⇨ 'c6235813-3ba4-3801-ae84-e0a6ebb7d138' // ... using a custom namespace // // Note: Custom namespaces should be a UUID string specific to your application! // E.g. the one here was generated using this modules `uuid` CLI. const MY\_NAMESPACE = '1b671a64-40d5-491e-99b0-da01ff1f3341'; uuidv3('Hello, World!', MY\_NAMESPACE); // ⇨ 'e8b5a51d-11c8-3310-a6ab-367563f20686' ``` Version 4 (random): ```javascript const uuidv4 = require('uuid/v4'); uuidv4(); // ⇨ '1b9d6bcd-bbfd-4b2d-9b5d-ab8dfbbd4bed' ``` Version 5 (namespace): ```javascript const uuidv5 = require('uuid/v5'); // ... using predefined DNS namespace (for domain names) uuidv5('hello.example.com', uuidv5.DNS); // ⇨ 'fdda765f-fc57-5604-a269-52a7df8164ec' // ... using predefined URL namespace (for, well, URLs) uuidv5('http://example.com/hello', uuidv5.URL); // ⇨ '3bbcee75-cecc-5b56-8031-b6641c1ed1f1' // ... using a custom namespace // // Note: Custom namespaces should be a UUID string specific to your application! // E.g. the one here was generated using this modules `uuid` CLI. const MY\_NAMESPACE = '1b671a64-40d5-491e-99b0-da01ff1f3341'; uuidv5('Hello, World!', MY\_NAMESPACE); // ⇨ '630eb68f-e0fa-5ecc-887a-7c7a62614681' ``` ## API ### Version 1 ```javascript const uuidv1 = require('uuid/v1'); // Incantations uuidv1(); uuidv1(options); uuidv1(options, buffer, offset); ``` Generate and return a RFC4122 v1 (timestamp-based) UUID. \* `options` - (Object) Optional uuid state to apply. Properties may include: \* `node` - (Array) Node id as Array of 6 bytes (per 4.1.6). Default: Randomly generated ID. See note 1. \* `clockseq` - (Number between 0 - 0x3fff) RFC clock sequence. Default: An internally maintained clockseq is used. \* `msecs` - (Number) Time in milliseconds since unix Epoch. Default: The current time is used. \* `nsecs` - (Number between 0-9999) additional time, in 100-nanosecond units. Ignored if `msecs` is unspecified. Default: internal uuid counter is used, as per 4.2.1.2. \* `buffer` - (Array | Buffer) Array or buffer where UUID bytes are to be written. \* `offset` - (Number) Starting index in `buffer` at which to begin writing. Returns `buffer`, if specified, otherwise the string form of the UUID Note: The default [node id](https://tools.ietf.org/html/rfc4122#section-4.1.6) (the last 12 digits in the UUID) is generated once, randomly, on process startup, and then remains unchanged for the duration of the process. Example: Generate string UUID with fully-specified options ```javascript const v1options = { node: [0x01, 0x23, 0x45, 0x67, 0x89, 0xab], clockseq: 0x1234, msecs: new Date('2011-11-01').getTime(), nsecs: 5678 }; uuidv1(v1options); // ⇨ '710b962e-041c-11e1-9234-0123456789ab' ``` Example: In-place generation of two binary IDs ```javascript // Generate two ids in an array const arr = new Array(); uuidv1(null, arr, 0); // ⇨ // [ // 44, 94, 164, 192, 64, 103, // 17, 233, 146, 52, 155, 29, // 235, 77, 59, 125 // ] uuidv1(null, arr, 16); // ⇨ // [ // 44, 94, 164, 192, 64, 103, 17, 233, // 146, 52, 155, 29, 235, 77, 59, 125, // 44, 94, 164, 193, 64, 103, 17, 233, // 146, 52, 155, 29, 235, 77, 59, 125 // ] ``` ### Version 3 ```javascript const uuidv3 = require('uuid/v3'); // Incantations uuidv3(name, namespace); uuidv3(name, namespace, buffer); uuidv3(name, namespace, buffer, offset); ``` Generate and return a RFC4122 v3 UUID. \* `name` - (String | Array[]) "name" to create UUID with \* `namespace` - (String | Array[]) "namespace" UUID either as a String or Array[16] of byte values \* `buffer` - (Array | Buffer) Array or buffer where UUID bytes are to be written. \* `offset` - (Number) Starting index in `buffer` at which to begin writing. Default = 0 Returns `buffer`, if specified, otherwise the string form of the UUID Example: ```javascript uuidv3('hello world', MY\_NAMESPACE); // ⇨ '042ffd34-d989-321c-ad06-f60826172424' ``` ### Version 4 ```javascript const uuidv4 = require('uuid/v4') // Incantations uuidv4(); uuidv4(options); uuidv4(options, buffer, offset); ``` Generate and return a RFC4122 v4 UUID. \* `options` - (Object) Optional uuid state to apply. Properties may include: \* `random` - (Number[16]) Array of 16 numbers (0-255) to use in place of randomly generated values \* `rng` - (Function) Random # generator function that returns an Array[16] of byte values (0-255) \* `buffer` - (Array | Buffer) Array or buffer where UUID bytes are to be written. \* `offset` - (Number) Starting index in `buffer` at which to begin writing. Returns `buffer`, if specified, otherwise the string form of the UUID Example: Generate string UUID with predefined `random` values ```javascript const v4options = { random: [ 0x10, 0x91, 0x56, 0xbe, 0xc4, 0xfb, 0xc1, 0xea, 0x71, 0xb4, 0xef, 0xe1, 0x67, 0x1c, 0x58, 0x36 ] }; uuidv4(v4options); // ⇨ '109156be-c4fb-41ea-b1b4-efe1671c5836' ``` Example: Generate two IDs in a single buffer ```javascript const buffer = new Array(); uuidv4(null, buffer, 0); // ⇨ // [ // 155, 29, 235, 77, 59, // 125, 75, 173, 155, 221, // 43, 13, 123, 61, 203, // 109 // ] uuidv4(null, buffer, 16); // ⇨ // [ // 155, 29, 235, 77, 59, 125, 75, 173, // 155, 221, 43, 13, 123, 61, 203, 109, // 27, 157, 107, 205, 187, 253, 75, 45, // 155, 93, 171, 141, 251, 189, 75, 237 // ] ``` ### Version 5 ```javascript const uuidv5 = require('uuid/v5'); // Incantations uuidv5(name, namespace); uuidv5(name, namespace, buffer); uuidv5(name, namespace, buffer, offset); ``` Generate and return a RFC4122 v5 UUID. \* `name` - (String | Array[]) "name" to create UUID with \* `namespace` - (String | Array[]) "namespace" UUID either as a String or Array[16] of byte values \* `buffer` - (Array | Buffer) Array or buffer where UUID bytes are to be written. \* `offset` - (Number) Starting index in `buffer` at which to begin writing. Default = 0 Returns `buffer`, if specified, otherwise the string form of the UUID Example: ```javascript uuidv5('hello world', MY\_NAMESPACE); // ⇨ '9f282611-e0fd-5650-8953-89c8e342da0b' ``` ## Command Line UUIDs can be generated from the command line with the `uuid` command. ```shell $ uuid ddeb27fb-d9a0-4624-be4d-4615062daed4 $ uuid v1 02d37060-d446-11e7-a9fa-7bdae751ebe1 ``` Type `uuid --help` for usage details ## Testing ```shell npm test ``` ---- Markdown generated from [README\_js.md](README\_js.md) by [![RunMD Logo](http://i.imgur.com/h0FVyzU.png)](https://github.com/broofa/runmd)