# uuid [![CI](https://github.com/uuidjs/uuid/workflows/CI/badge.svg)](https://github.com/uuidjs/uuid/actions?query=workflow%3ACI) [![Browser](https://github.com/uuidjs/uuid/workflows/Browser/badge.svg)](https://github.com/uuidjs/uuid/actions?query=workflow%3ABrowser) For the creation of [RFC9562](https://www.rfc-editor.org/rfc/rfc9562.html) (formally [RFC4122](https://www.rfc-editor.org/rfc/rfc4122.html)) UUIDs - \*\*Complete\*\* - Support for all RFC9562 (nee RFC4122) UUID versions - \*\*Cross-platform\*\* - Support for ... - CommonJS, [ECMAScript Modules](#ecmascript-modules) and [CDN builds](#cdn-builds) - NodeJS 16+ ([LTS releases](https://github.com/nodejs/Release)) - Chrome, Safari, Firefox, Edge browsers - Webpack and rollup.js module bundlers - [React Native / Expo](#react-native--expo) - \*\*Secure\*\* - Cryptographically-strong random values - \*\*Small\*\* - Zero-dependency, small footprint, plays nice with "tree shaking" packagers - \*\*CLI\*\* - Includes the [`uuid` command line](#command-line) utility > [!NOTE] > Upgrading from `uuid@3`? Your code is probably okay, but check out [Upgrading From `uuid@3`](#upgrading-from-uuid3) for details. > [!NOTE] > Only interested in creating a version 4 UUID? You might be able to use [`crypto.randomUUID()`](https://developer.mozilla.org/en-US/docs/Web/API/Crypto/randomUUID), eliminating the need to install this library. ## Quickstart To create a random UUID... \*\*1. Install\*\* ```shell npm install uuid ``` \*\*2. Create a UUID\*\* (ES6 module syntax) ```javascript import { v4 as uuidv4 } from 'uuid'; uuidv4(); // ⇨ '9b1deb4d-3b7d-4bad-9bdd-2b0d7b3dcb6d' ``` ... or using CommonJS syntax: ```javascript const { v4: uuidv4 } = require('uuid'); uuidv4(); // ⇨ '1b9d6bcd-bbfd-4b2d-9b5d-ab8dfbbd4bed' ``` For timestamp UUIDs, namespace UUIDs, and other options read on ... ## API Summary | | | | | --- | --- | --- | | [`uuid.NIL`](#uuidnil) | The nil UUID string (all zeros) | New in `uuid@8.3` | | [`uuid.MAX`](#uuidmax) | The max UUID string (all ones) | New in `uuid@9.1` | | [`uuid.parse()`](#uuidparsestr) | Convert UUID string to array of bytes | New in `uuid@8.3` | | [`uuid.stringify()`](#uuidstringifyarr-offset) | Convert array of bytes to UUID string | New in `uuid@8.3` | | [`uuid.v1()`](#uuidv1options-buffer-offset) | Create a version 1 (timestamp) UUID | | | [`uuid.v1ToV6()`](#uuidv1tov6uuid) | Create a version 6 UUID from a version 1 UUID | New in `uuid@10` | | [`uuid.v3()`](#uuidv3name-namespace-buffer-offset) | Create a version 3 (namespace w/ MD5) UUID | | | [`uuid.v4()`](#uuidv4options-buffer-offset) | Create a version 4 (random) UUID | | | [`uuid.v5()`](#uuidv5name-namespace-buffer-offset) | Create a version 5 (namespace w/ SHA-1) UUID | | | [`uuid.v6()`](#uuidv6options-buffer-offset) | Create a version 6 (timestamp, reordered) UUID | New in `uuid@10` | | [`uuid.v6ToV1()`](#uuidv6tov1uuid) | Create a version 1 UUID from a version 6 UUID | New in `uuid@10` | | [`uuid.v7()`](#uuidv7options-buffer-offset) | Create a version 7 (Unix Epoch time-based) UUID | New in `uuid@10` | | ~~[`uuid.v8()`](#uuidv8)~~ | "Intentionally left blank" | | | [`uuid.validate()`](#uuidvalidatestr) | Test a string to see if it is a valid UUID | New in `uuid@8.3` | | [`uuid.version()`](#uuidversionstr) | Detect RFC version of a UUID | New in `uuid@8.3` | ## API ### uuid.NIL The nil UUID string (all zeros). Example: ```javascript import { NIL as NIL\_UUID } from 'uuid'; NIL\_UUID; // ⇨ '00000000-0000-0000-0000-000000000000' ``` ### uuid.MAX The max UUID string (all ones). Example: ```javascript import { MAX as MAX\_UUID } from 'uuid'; MAX\_UUID; // ⇨ 'ffffffff-ffff-ffff-ffff-ffffffffffff' ``` ### uuid.parse(str) Convert UUID string to array of bytes | | | | --------- | ---------------------------------------- | | `str` | A valid UUID `String` | | \_returns\_ | `Uint8Array[16]` | | \_throws\_ | `TypeError` if `str` is not a valid UUID | > [!NOTE] > Ordering of values in the byte arrays used by `parse()` and `stringify()` follows the left ↠ right order of hex-pairs in UUID strings. As shown in the example below. Example: ```javascript import { parse as uuidParse } from 'uuid'; // Parse a UUID const bytes = uuidParse('6ec0bd7f-11c0-43da-975e-2a8ad9ebae0b'); // Convert to hex strings to show byte order (for documentation purposes) [...bytes].map((v) => v.toString(16).padStart(2, '0')); // ⇨ // [ // '6e', 'c0', 'bd', '7f', // '11', 'c0', '43', 'da', // '97', '5e', '2a', '8a', // 'd9', 'eb', 'ae', '0b' // ] ``` ### uuid.stringify(arr[, offset]) Convert array of bytes to UUID string | | | | -------------- | ---------------------------------------------------------------------------- | | `arr` | `Array`-like collection of 16 values (starting from `offset`) between 0-255. | | [`offset` = 0] | `Number` Starting index in the Array | | \_returns\_ | `String` | | \_throws\_ | `TypeError` if a valid UUID string cannot be generated | > [!NOTE] > Ordering of values in the byte arrays used by `parse()` and `stringify()` follows the left ↠ right order of hex-pairs in UUID strings. As shown in the example below. Example: ```javascript import { stringify as uuidStringify } from 'uuid'; const uuidBytes = [ 0x6e, 0xc0, 0xbd, 0x7f, 0x11, 0xc0, 0x43, 0xda, 0x97, 0x5e, 0x2a, 0x8a, 0xd9, 0xeb, 0xae, 0x0b, ]; uuidStringify(uuidBytes); // ⇨ '6ec0bd7f-11c0-43da-975e-2a8ad9ebae0b' ``` ### uuid.v1([options[, buffer[, offset]]]) Create an RFC version 1 (timestamp) UUID | | | | --- | --- | | [`options`] | `Object` with one or more of the following properties: | | [`options.node` ] | RFC "node" field as an `Array[6]` of byte values (per 4.1.6) | | [`options.clockseq`] | RFC "clock sequence" as a `Number` between 0 - 0x3fff | | [`options.msecs`] | RFC "timestamp" field (`Number` of milliseconds, unix epoch) | | [`options.nsecs`] | RFC "timestamp" field (`Number` of nanoseconds to add to `msecs`, should be 0-10,000) | | [`options.random`] | `Array` of 16 random bytes (0-255) | | [`options.rng`] | Alternative to `options.random`, a `Function` that returns an `Array` of 16 random bytes (0-255) | | [`buffer`] | `Array \| Buffer` If specified, uuid will be written here in byte-form, starting at `offset` | | [`offset` = 0] | `Number` Index to start writing UUID bytes in `buffer` | | \_returns\_ | UUID `String` if no `buffer` is specified, otherwise returns `buffer` | | \_throws\_ | `Error` if more than 10M UUIDs/sec are requested | > [!NOTE] > The default [node id](https://datatracker.ietf.org/doc/html/rfc9562#section-5.1) (the last 12 digits in the UUID) is generated once, randomly, on process startup, and then remains unchanged for the duration of the process. > [!NOTE] > `options.random` and `options.rng` are only meaningful on the very first call to `v1()`, where they may be passed to initialize the internal `node` and `clockseq` fields. Example: ```javascript import { v1 as uuidv1 } from 'uuid'; uuidv1(); // ⇨ '2c5ea4c0-4067-11e9-8bad-9b1deb4d3b7d' ``` Example using `options`: ```javascript import { v1 as uuidv1 } from 'uuid'; const options = { node: [0x01, 0x23, 0x45, 0x67, 0x89, 0xab], clockseq: 0x1234, msecs: new Date('2011-11-01').getTime(), nsecs: 5678, }; uuidv1(options); // ⇨ '710b962e-041c-11e1-9234-0123456789ab' ``` ### uuid.v1ToV6(uuid) Convert a UUID from version 1 to version 6 ```javascript import { v1ToV6 } from 'uuid'; v1ToV6('92f62d9e-22c4-11ef-97e9-325096b39f47'); // ⇨ '1ef22c49-2f62-6d9e-97e9-325096b39f47' ``` ### uuid.v3(name, namespace[, buffer[, offset]]) Create an RFC version 3 (namespace w/ MD5) UUID API is identical to `v5()`, but uses "v3" instead. > [!IMPORTANT] > Per the RFC, "\_If backward compatibility is not an issue, SHA-1 [Version 5] is preferred\_." ### uuid.v4([options[, buffer[, offset]]]) Create an RFC version 4 (random) UUID | | | | --- | --- | | [`options`] | `Object` with one or more of the following properties: | | [`options.random`] | `Array` of 16 random bytes (0-255) | | [`options.rng`] | Alternative to `options.random`, a `Function` that returns an `Array` of 16 random bytes (0-255) | | [`buffer`] | `Array \| Buffer` If specified, uuid will be written here in byte-form, starting at `offset` | | [`offset` = 0] | `Number` Index to start writing UUID bytes in `buffer` | | \_returns\_ | UUID `String` if no `buffer` is specified, otherwise returns `buffer` | Example: ```javascript import { v4 as uuidv4 } from 'uuid'; uuidv4(); // ⇨ '1b9d6bcd-bbfd-4b2d-9b5d-ab8dfbbd4bed' ``` Example using predefined `random` values: ```javascript import { v4 as uuidv4 } from 'uuid'; 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' ``` ### uuid.v5(name, namespace[, buffer[, offset]]) Create an RFC version 5 (namespace w/ SHA-1) UUID | | | | --- | --- | | `name` | `String \| Array` | | `namespace` | `String \| Array[16]` Namespace UUID | | [`buffer`] | `Array \| Buffer` If specified, uuid will be written here in byte-form, starting at `offset` | | [`offset` = 0] | `Number` Index to start writing UUID bytes in `buffer` | | \_returns\_ | UUID `String` if no `buffer` is specified, otherwise returns `buffer` | > [!NOTE] > The RFC `DNS` and `URL` namespaces are available as `v5.DNS` and `v5.URL`. Example with custom namespace: ```javascript import { v5 as uuidv5 } from 'uuid'; // Define a custom namespace. Readers, create your own using something like // https://www.uuidgenerator.net/ const MY\_NAMESPACE = '1b671a64-40d5-491e-99b0-da01ff1f3341'; uuidv5('Hello, World!', MY\_NAMESPACE); // ⇨ '630eb68f-e0fa-5ecc-887a-7c7a62614681' ``` Example with RFC `URL` namespace: ```javascript import { v5 as uuidv5 } from 'uuid'; uuidv5('https://www.w3.org/', uuidv5.URL); // ⇨ 'c106a26a-21bb-5538-8bf2-57095d1976c1' ``` ### uuid.v6([options[, buffer[, offset]]]) Create an RFC version 6 (timestamp, reordered) UUID This method takes the same arguments as uuid.v1(). ```javascript import { v6 as uuidv6 } from 'uuid'; uuidv6(); // ⇨ '1e940672-c5ea-64c0-8bad-9b1deb4d3b7d' ``` Example using `options`: ```javascript import { v6 as uuidv6 } from 'uuid'; const options = { node: [0x01, 0x23, 0x45, 0x67, 0x89, 0xab], clockseq: 0x1234, msecs: new Date('2011-11-01').getTime(), nsecs: 5678, }; uuidv6(options); // ⇨ '1e1041c7-10b9-662e-9234-0123456789ab' ``` ### uuid.v6ToV1(uuid) Convert a UUID from version 6 to version 1 ```javascript import { v6ToV1 } from 'uuid'; v6ToV1('1ef22c49-2f62-6d9e-97e9-325096b39f47'); // ⇨ '92f62d9e-22c4-11ef-97e9-325096b39f47' ``` ### uuid.v7([options[, buffer[, offset]]]) Create an RFC version 7 (random) UUID | | | | --- | --- | | [`options`] | `Object` with one or more of the following properties: | | [`options.msecs`] | RFC "timestamp" field (`Number` of milliseconds, unix epoch) | | [`options.random`] | `Array` of 16 random bytes (0-255) | | [`options.rng`] | Alternative to `options.random`, a `Function` that returns an `Array` of 16 random bytes (0-255) | | [`options.seq`] | 31 bit monotonic sequence counter as `Number` between 0 - 0x7fffffff | | [`buffer`] | `Array \| Buffer` If specified, uuid will be written here in byte-form, starting at `offset` | | [`offset` = 0] | `Number` Index to start writing UUID bytes in `buffer` | | \_returns\_ | UUID `String` if no `buffer` is specified, otherwise returns `buffer` | Example: ```javascript import { v7 as uuidv7 } from 'uuid'; uuidv7(); // ⇨ '01695553-c90c-722d-9b5d-b38dfbbd4bed' ``` ### ~~uuid.v8()~~ \*\*\_"Intentionally left blank"\_\*\* > [!NOTE] > Version 8 (experimental) UUIDs are "[for experimental or vendor-specific use cases](https://www.rfc-editor.org/rfc/rfc9562.html#name-uuid-version-8)". The RFC does not define a creation algorithm for them, which is why this package does not offer a `v8()` method. The `validate()` and `version()` methods do work with such UUIDs, however. ### uuid.validate(str) Test a string to see if it is a valid UUID | | | | --------- | --------------------------------------------------- | | `str` | `String` to validate | | \_returns\_ | `true` if string is a valid UUID, `false` otherwise | Example: ```javascript import { validate as uuidValidate } from 'uuid'; uuidValidate('not a UUID'); // ⇨ false uuidValidate('6ec0bd7f-11c0-43da-975e-2a8ad9ebae0b'); // ⇨ true ``` Using `validate` and `version` together it is possible to do per-version validation, e.g. validate for only v4 UUIds. ```javascript import { version as uuidVersion } from 'uuid'; import { validate as uuidValidate } from 'uuid'; function uuidValidateV4(uuid) { return uuidValidate(uuid) && uuidVersion(uuid) === 4; } const v1Uuid = 'd9428888-122b-11e1-b85c-61cd3cbb3210'; const v4Uuid = '109156be-c4fb-41ea-b1b4-efe1671c5836'; uuidValidateV4(v4Uuid); // ⇨ true uuidValidateV4(v1Uuid); // ⇨ false ``` ### uuid.version(str) Detect RFC version of a UUID | | | | --------- | ---------------------------------------- | | `str` | A valid UUID `String` | | \_returns\_ | `Number` The RFC version of the UUID | | \_throws\_ | `TypeError` if `str` is not a valid UUID | Example: ```javascript import { version as uuidVersion } from 'uuid'; uuidVersion('45637ec4-c85f-11ea-87d0-0242ac130003'); // ⇨ 1 uuidVersion('6ec0bd7f-11c0-43da-975e-2a8ad9ebae0b'); // ⇨ 4 ``` > [!NOTE] > This method returns `0` for the `NIL` UUID, and `15` for the `MAX` UUID. ## Command Line UUIDs can be generated from the command line using `uuid`. ```shell $ npx uuid ddeb27fb-d9a0-4624-be4d-4615062daed4 ``` The default is to generate version 4 UUIDS, however the other versions are supported. Type `uuid --help` for details: ```shell $ npx uuid --help Usage: uuid uuid v1 uuid v3 uuid v4 uuid v5 uuid v7 uuid --help Note: may be "URL" or "DNS" to use the corresponding UUIDs defined by RFC9562 ``` ## ECMAScript Modules This library comes with [ECMAScript Modules](https://www.ecma-international.org/ecma-262/6.0/#sec-modules) (ESM) support for Node.js versions that support it ([example](./examples/node-esmodules/)) as well as bundlers like [rollup.js](https://rollupjs.org/guide/en/#tree-shaking) ([example](./examples/browser-rollup/)) and [webpack](https://webpack.js.org/guides/tree-shaking/) ([example](./examples/browser-webpack/)) (targeting both, Node.js and browser environments). ```javascript import { v4 as uuidv4 } from 'uuid'; uuidv4(); // ⇨ '1b9d6bcd-bbfd-4b2d-9b5d-ab8dfbbd4bed' ``` To run the examples you must first create a dist build of this library in the module root: ```shell npm run build ``` ## CDN Builds ### ECMAScript Modules To load this module directly into modern browsers that [support loading ECMAScript Modules](https://caniuse.com/#feat=es6-module) you can make use of [jspm](https://jspm.org/): ```html ``` ### UMD As of `uuid@9` [UMD (Universal Module Definition)](https://github.com/umdjs/umd) builds are no longer shipped with this library. If you need a UMD build of this library, use a bundler like Webpack or Rollup. Alternatively, refer to the documentation of [`uuid@8.3.2`](https://github.com/uuidjs/uuid/blob/v8.3.2/README.md#umd) which was the last version that shipped UMD builds. ## Known issues ### Duplicate UUIDs (Googlebot) This module may generate duplicate UUIDs when run in clients with \_deterministic\_ random number generators, such as [Googlebot crawlers](https://developers.google.com/search/docs/advanced/crawling/overview-google-crawlers). This can cause problems for apps that expect client-generated UUIDs to always be unique. Developers should be prepared for this and have a strategy for dealing with possible collisions, such as: - Check for duplicate UUIDs, fail gracefully - Disable write operations for Googlebot clients ### "getRandomValues() not supported" This error occurs in environments where the standard [`crypto.getRandomValues()`](https://developer.mozilla.org/en-US/docs/Web/API/Crypto/getRandomValues) API is not supported. This issue can be resolved by adding an appropriate polyfill: ### React Native / Expo 1. Install [`react-native-get-random-values`](https://github.com/LinusU/react-native-get-random-values#readme) 1. Import it \_before\_ `uuid`. Since `uuid` might also appear as a transitive dependency of some other imports it's safest to just import `react-native-get-random-values` as the very first thing in your entry point: ```javascript import 'react-native-get-random-values'; import { v4 as uuidv4 } from 'uuid'; ``` > [!NOTE] > If you are using Expo, you must be using at least `react-native-get-random-values@1.5.0` and `expo@39.0.0`. ### Web Workers / Service Workers (Edge <= 18) [In Edge <= 18, Web Crypto is not supported in Web Workers or Service Workers](https://caniuse.com/#feat=cryptography) and we are not aware of a polyfill (let us know if you find one, please). ### IE 11 (Internet Explorer) Support for IE11 and other legacy browsers has been dropped as of `uuid@9`. If you need to support legacy browsers, you can always transpile the uuid module source yourself (e.g. using [Babel](https://babeljs.io/)). ## Upgrading From `uuid@7` ### Only Named Exports Supported When Using with Node.js ESM `uuid@7` did not come with native ECMAScript Module (ESM) support for Node.js. Importing it in Node.js ESM consequently imported the CommonJS source with a default export. This library now comes with true Node.js ESM support and only provides named exports. Instead of doing: ```javascript import uuid from 'uuid'; uuid.v4(); ``` you will now have to use the named exports: ```javascript import { v4 as uuidv4 } from 'uuid'; uuidv4(); ``` ### Deep Requires No Longer Supported Deep requires like `require('uuid/v4')` [which have been deprecated in `uuid@7`](#deep-requires-now-deprecated) are no longer supported. ## Upgrading From `uuid@3` "\_Wait... what happened to `uuid@4` thru `uuid@6`?!?\_" In order to avoid confusion with RFC [version 4](#uuidv4options-buffer-offset) and [version 5](#uuidv5name-namespace-buffer-offset) UUIDs, and a possible [version 6](http://gh.peabody.io/uuidv6/), releases 4 thru 6 of this module have been skipped. ### Deep Requires Now Deprecated `uuid@3` encouraged the use of deep requires to minimize the bundle size of browser builds: ```javascript const uuidv4 = require('uuid/v4'); // <== NOW DEPRECATED! uuidv4(); ``` As of `uuid@7` this library now provides ECMAScript modules builds, which allow packagers like Webpack and Rollup to do "tree-shaking" to remove dead code. Instead, use the `import` syntax: ```javascript import { v4 as uuidv4 } from 'uuid'; uuidv4(); ``` ... or for CommonJS: ```javascript const { v4: uuidv4 } = require('uuid'); uuidv4(); ``` ### Default Export Removed `uuid@3` was exporting the Version 4 UUID method as a default export: ```javascript const uuid = require('uuid'); // <== REMOVED! ``` This usage pattern was already discouraged in `uuid@3` and has been removed in `uuid@7`. --- Markdown generated from [README\_js.md](README\_js.md) by 