Create GUID / UUID in JavaScript?

3745

1263

I'm trying to create globally-unique identifiers in JavaScript. I'm not sure what routines are available on all browsers, how "random" and seeded the built-in random number generator is, etc..

The GUID / UUID should be at least 32 characters and should stay in the ASCII range to avoid trouble when passing them around.

javascript guid uuid

edited Dec 26 '14 at 23:36

community wiki
7 revs, 5 users 50%
Jason Cohen

10 GUIDs when repesented as as strings are at least 36 and no more than 38 characters in length and match the pattern ^\{?[a-zA-Z0-9]{36}?\}$ and hence are always ascii. – AnthonyWJones Sep 19 '08 at 20:35

1 David Bau provides a much better, seedable random number generator at davidbau.com/archives/2010/01/30/… I wrote up a slightly different approach to generating UUIDs at blogs.cozi.com/tech/2010/04/generating-uuids-in-javascript.html – George V. Reilly May 4 '10 at 23:09

4 jsben.ch/#/Lbxoe - here a benchmark with the different functions from below – EscapeNetscape Oct 24 '16 at 17:59

2 uuid-random uses good PRNG and is very fast. – jchook Oct 29 '16 at 16:37

3 Late (very late!) to the party here but @AnthonyWJones shouldn't your regex read: ^\{?[a-fA-F0-9]{36}?\}$ – noonand Feb 13 '17 at 11:40

52 Answers

1 2 next

UUIDs (Universally Unique IDentifier), also known as GUIDs (Globally Unique IDentifier), according to RFC 4122, are identifiers with a certain uniqueness guarantee

A popular Open Source tool for working with UUIDs in JavaScript is node-uuid

Note that just randomly generating the identifiers byte by byte, or character by character, will not give you the same guarantees as a conforming implementation. Also, very important, systems working with compliant UUIDs may choose not to accept randomly generated ones, and many open source validators will actually check for a valid structure.

An UUID must have this format:

```
xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx
```

Where the *M* and *N* positions may only have certain values. At this time, the only valid values for M are 1, 2, 3, 4 and 5, so randomly generating that position would make most results unacceptable.

edited May 22 at 14:01

community wiki
21 revs, 16 users 17%
Jon Surrell

172 Actually, the RFC allows for UUIDs that are created from random numbers. You just have to twiddle a couple of bits to identify it as such. See section 4.4. Algorithms for Creating a UUID from Truly Random or Pseudo-Random Numbers: rfc-archive.org/getrfc.php?rfc=4122 – Jason DeFontes Sep 19 '08 at 20:28

4 Could someone explain this code to me? It looks like the S4 function tries to get a random hex number between 0x10000 and 0x20000, then outputs the last 4 digits. But why the bitwise or with 0? Shouldn't that be a noop? Also, is the 0x10000 to 0x20000 just a hack to avoid having to deal with leading 0's? – Cory Oct 26 '10 at 22:56

17 In Chrome this code doesn't always generate a correct size GUID. Length varies between 35 and 36 – cdeutsch Sep 13 '12 at 21:38

182 How can a so obviously wrong answer get so many upvotes? Even the code is wrong, as there is not a 4 at the right position. en.wikipedia.org/wiki/Globally_unique_identifier – Dennis Krøger Jan 21 '13 at 9:28

14 This answer was broken in revision 5 when "1+...." and "substring(1)" were removed. Those bits guaranteed the consistent length. – Segfault Feb 7 '13 at 19:12

For an RFC4122 version 4 compliant solution, this one-liner(ish) solution is the most compact I could come up with.:

```
  });
}

console.log(uuidv4())
```

Run code snippet Expand snippet

Update, 2015-06-02: Be aware that UUID uniqueness relies heavily on the underlying random number generator (RNG). The solution above uses `Math.random()` for brevity, however `Math.random()` is *not* guaranteed to be a high-quality RNG. See Adam Hyland's excellent writeup on Math.random() for details. For a more robust solution, consider something like the uuid module[Disclaimer: I'm the author], which uses higher quality RNG APIs where available.

Update, 2015-08-26: As a side-note, this gist describes how to determine how many IDs can be generated before reaching a certain probability of collision. For example, with 3.26×10^{15} version 4 RFC4122 UUIDs you have a 1-in-a-million chance of collision.

Update, 2017-06-28: A good article from Chrome developers discussing the state of Math.random PRNG quality in Chrome, Firefox, and Safari. tl;dr - As of late-2015 it's "pretty good", but not cryptographic quality. To address that issue, here's an updated version of the above solution that uses ES6, the `crypto` API, and a bit of JS wizardy I can't take credit for:

```
function uuidv4() {
  return ([1e7]+-1e3+-4e3+-8e3+-1e11).replace(/[018]/g, c =>
    (c ^ crypto.getRandomValues(new Uint8Array(1))[0] & 15 >> c / 4).toString(16)
  )
}

console.log(uuidv4());
```

Run code snippet Expand snippet

edited May 25 at 12:50

community wiki
13 revs, 5 users 58%
broofa

30 I posted a question about collisions stackoverflow.com/questions/6906916/… – Muxa Aug 2 '11 at 3:23

126 Surely the answer to @Muxa's question is 'no'? It's never truly safe to trust something that came from the client. I guess it depends on how likely your users are to bring up a javascript console and manually change the variable so to something they want. Or they could just POST you back the id that they want. It would also depend on whether the user picking their own ID is going to cause vulnerabilities. Either way, if it's a random number ID that's going into a table, I would probably be generating it server-side, so that I know I have control over the process. – Cam Jackson Nov 1 '12 at 14:34

34 @DrewNoakes - UUIDs aren't just a string of completely random #'s. The "4" is the uuid version (4 = "random"). The "y" marks where the uuid variant (field layout, basically) needs to be embedded. See sections 4.1.1 and 4.1.3 of ietf.org/rfc/rfc4122.txt for more info. – broofa Nov 27 '12 at 22:13

755

I really like how clean Broofa's answer is, but it's unfortunate that poor implementations of `Math.random` leave the chance for collision.

Here's a similar RFC4122 version 4 compliant solution that solves that issue by offsetting the first 13 hex numbers by a hex portion of the timestamp. That way, even if `Math.random` is on the same seed, both clients would have to generate the UUID at the exact same millisecond (or 10,000+ years later) to get the same UUID:

```
function generateUUID() { // Public Domain/MIT
    var d = new Date().getTime();
    if (typeof performance !== 'undefined' && typeof performance.now === 'function'){
        d += performance.now(); //use high-precision timer if available
    }
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function (c) {
        var r = (d + Math.random() * 16) % 16 | 0;
        d = Math.floor(d / 16);
        return (c === 'x' ? r : (r & 0x3 | 0x8)).toString(16);
    });
}
```

Here's a fiddle to test.

edited Apr 2 '18 at 18:34

community wiki
14 revs, 7 users 80%
Briguy37

`Date.now()` to `new Date().getTime()` – Fresheyeball Jun 28 '13 at 19:47

76 performance.now would be even better. Unlike Date.now, the timestamps returned by `performance.now()` are not limited to one-millisecond resolution. Instead, they represent times as floating-point numbers with up to **microsecond precision**. Also unlike Date.now, the values returned by performance.now() **always increase at a constant rate**, independent of the system clock which might be adjusted manually or skewed by software such as the Network Time Protocol. – daniellmb Mar 13 '14 at 4:25

6 @daniellmb You probably should've linked to MDN or another to show real documentation and not a polyfill ;) – Martin Jul 8 '14 at 19:38

11 FYI, per the site footer, all user contributions on the site are available under the cc by-sa 3.0 license. – Xiong Chiamiov Feb 4 '15 at 1:34

373

broofa's answer is pretty slick, indeed - impressively clever, really... rfc4122 compliant, somewhat readable, and compact. Awesome!

But if you're looking at that regular expression, those many `replace()` callbacks, `toString()` 's and `Math.random()` function calls (where he's only using 4 bits of the result and wasting the rest), you may start to wonder about performance. Indeed, joelpt even decided to toss out RFC for generic GUID speed with `generateQuickGUID` .

But, can we get speed *and* RFC compliance? I say, YES! Can we maintain readability? Well... Not really, but it's easy if you follow along.

But first, my results, compared to broofa, `guid` (the accepted answer), and the non-rfc-compliant `generateQuickGuid` :

```
                  Desktop   Android
            broofa: 1617ms   12869ms
                e1:  636ms    5778ms
                e2:  606ms    4754ms
                e3:  364ms    3003ms
                e4:  329ms    2015ms
                e5:  147ms    1156ms
                e6:  146ms    1035ms
                e7:  105ms     726ms
              guid:  962ms   10762ms
generateQuickGuid:  292ms    2961ms
  - Note: 500k iterations, results will vary by browser/cpu.
```

So by my 6th iteration of optimizations, I beat the most popular answer by over **12X**, the accepted answer by over **9X**, and the fast-non-compliant answer by **2.3X**. And I'm still rfc4122 compliant.

```
'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
  var r = Math.random()*16|0, v = c == 'x' ? r : (r&0x3|0x8);
  return v.toString(16);
});
```

So it replaces `x` with any random hex digit, `y` with random data (except forcing the top 2 bits to `10` per the RFC spec), and the regex doesn't match the `-` or `4` characters, so he doesn't have to deal with them. Very, very slick.

The first thing to know is that function calls are expensive, as are regular expressions (though he only uses 1, it has 32 callbacks, one for each match, and in each of the 32 callbacks it calls Math.random() and v.toString(16)).

The first step toward performance is to eliminate the RegEx and its callback functions and use a simple loop instead. This means we have to deal with the `-` and `4` characters whereas broofa did not. Also, note that we can use String Array indexing to keep his slick String template architecture:

```
function e1() {
  var u='',i=0;
  while(i++<36) {
    var c='xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'[i-1],r=Math.random()*16|0,v=c=='x'?r:
(r&0x3|0x8);
    u+=(c=='-'||c=='4')?c:v.toString(16)
  }
  return u;
}
```

Basically, the same inner logic, except we check for `-` or `4`, and using a while loop (instead of `replace()` callbacks) gets us an almost 3X improvement!

The next step is a small one on the desktop but makes a decent difference on mobile. Let's make fewer Math.random() calls and utilize all those random bits instead of throwing 87% of them away with a random buffer that gets shifted out each iteration. Let's also move that template definition out of the loop, just in case it helps:

```
function e2() {
  var u='',m='xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx',i=0,rb=Math.random()*0xffffffff|0;
  while(i++<36) {
    var c=m[i-1],r=rb&0xf,v=c=='x'?r:(r&0x3|0x8);
```

This saves us 10-30% depending on platform. Not bad. But the next big step gets rid of the toString function calls altogether with an optimization classic - the look-up table. A simple 16-element lookup table will perform the job of toString(16) in much less time:

```
function e3() {
  var h='0123456789abcdef';
  var k='xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx';
  /* same as e4() below */
}
function e4() {
  var h=['0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'];
  var k=['x','x','x','x','x','x','x','x','-','x','x','x','x','-','4','x','x','x','-
','y','x','x','x','-','x','x','x','x','x','x','x','x','x','x','x','x'];
  var u='',i=0,rb=Math.random()*0xffffffff|0;
  while(i++<36) {
    var c=k[i-1],r=rb&0xf,v=c=='x'?r:(r&0x3|0x8);
    u+=(c=='-'||c=='4')?c:h[v];rb=i%8==0?Math.random()*0xffffffff|0:rb>>4
  }
  return u
}
```

The next optimization is another classic. Since we're only handling 4-bits of output in each loop iteration, let's cut the number of loops in half and process 8-bits each iteration. This is tricky since we still have to handle the RFC compliant bit positions, but it's not too hard. We then have to make a larger lookup table (16x16, or 256) to store 0x00 - 0xff, and we build it only once, outside the e5() function.

```
var lut = []; for (var i=0; i<256; i++) { lut[i] = (i<16?'0':'')+(i).toString(16); }
function e5() {
  var k=['x','x','x','x','-','x','x','-','4','x','-','y','x','-
','x','x','x','x','x','x'];
  var u='',i=0,rb=Math.random()*0xffffffff|0;
  while(i++<20) {
    var c=k[i-1],r=rb&0xff,v=c=='x'?r:(c=='y'?(r&0x3f|0x80):(r&0xf|0x40));
    u+=(c=='-')?c:lut[v];rb=i%4==0?Math.random()*0xffffffff|0:rb>>8
  }
  return u
}
```

I tried an e6() that processes 16-bits at a time, still using the 256-element LUT, and it showed the diminishing returns of optimization. Though it had fewer iterations, the inner logic was complicated by the increased processing, and it performed the same on desktop, and

```
var lut = []; for (var i=0; i<256; i++) { lut[i] = (i<16?'0':'')+(i).toString(16); }
function e7()
{
  var d0 = Math.random()*0xffffffff|0;
  var d1 = Math.random()*0xffffffff|0;
  var d2 = Math.random()*0xffffffff|0;
  var d3 = Math.random()*0xffffffff|0;
  return lut[d0&0xff]+lut[d0>>8&0xff]+lut[d0>>16&0xff]+lut[d0>>24&0xff]+'-'+
    lut[d1&0xff]+lut[d1>>8&0xff]+'-'+lut[d1>>16&0x0f|0x40]+lut[d1>>24&0xff]+'-'+
    lut[d2&0x3f|0x80]+lut[d2>>8&0xff]+'-'+lut[d2>>16&0xff]+lut[d2>>24&0xff]+
    lut[d3&0xff]+lut[d3>>8&0xff]+lut[d3>>16&0xff]+lut[d3>>24&0xff];
}
```

Modualized: http://jcward.com/UUID.js - `UUID.generate()`

The funny thing is, generating 16 bytes of random data is the easy part. The whole trick is expressing it in String format with RFC compliance, and it's most tightly accomplished with 16 bytes of random data, an unrolled loop and lookup table.

I hope my logic is correct -- it's very easy to make a mistake in this kind of tedious bit-work. But the outputs look good to me. I hope you enjoyed this mad ride through code optimization!

Be advised: my primary goal was to show and teach potential optimization strategies. Other answers cover important topics such as collisions and truly random numbers, which are important for generating good UUIDs.

edited Aug 3 '18 at 19:45

community wiki
11 revs, 2 users 96%
Jeff Ward

1 jsperf.com would allow you to capture the data and see the results across browsers and devices. – fearphage Feb 24 '14 at 15:40

2 Hi @chad, good questions. k could be moved outside, but that didn't improve performance above and makes scope messier. And building an array and joining on return oddly kills performance. But again, feel free to experiment! – Jeff Ward Feb 24 '14 at 19:23

3 I'd be curious to see how node-uuid.js compares. In my work there I started out with more of a focus on performance, some of which remains. But I've since backed away from that, preferring to have more readable/maintainable code. The reason being that uuid perf is simply not an issue in the real world. Uuids are typically created in conjunction with much slower operations (e.g. making a network request, creating and persisting a model object), where shaving a few microseconds off things simply doesn't matter. – broofa Jun 2 '15 at 15:14

7 Have applied all the advices of @Dave and published very neat ES6/Babel source here: codepen.io/avesus/pen/wgQmaV?editors=0012 – Brian Haak Feb 10 '17 at 20:51

148

Here's some code based on RFC 4122, section 4.4 (Algorithms for Creating a UUID from Truly Random or Pseudo-Random Number).

```
function createUUID() {
    // http://www.ietf.org/rfc/rfc4122.txt
    var s = [];
    var hexDigits = "0123456789abcdef";
    for (var i = 0; i < 36; i++) {
        s[i] = hexDigits.substr(Math.floor(Math.random() * 0x10), 1);
    }
    s[14] = "4";  // bits 12-15 of the time_hi_and_version field to 0010
    s[19] = hexDigits.substr((s[19] & 0x3) | 0x8, 1);  // bits 6-7 of the
clock_seq_hi_and_reserved to 01
    s[8] = s[13] = s[18] = s[23] = "-";

    var uuid = s.join("");
    return uuid;
}
```

edited Oct 25 '11 at 22:37

community wiki
Kevin Hakanson

3 This doesn't produce the dashes needed for c# to parse it into a System.Guid. It renders like this: B42A153F1D9A4F92990392C11DD684D2, when it should render like: B42A153F-1D9A-4F92-9903-92C11DD684D2 – Levitikon Oct 25 '11 at 16:24

5 The ABNF from the spec does include the "-" characters, so I updated to be compliant. – Kevin Hakanson Oct 25 '11 at 22:40

17 I personally hate the dashes, but to each their own. Hey that's why we're programmers! – devios1 Jan 23 '12 at 16:20

4 You should declare the array size beforehand rather than sizing it dynamically as you build the GUID. `var s = new Array(36);` – MgSam Mar 25 '13 at 20:03

9 @Levitikon .NET's `Guid.Parse()` should parse `B42A153F1D9A4F92990392C11DD684D2` into a Guid just fine. It doesn't need to have the hyphens. – JLRishe Nov 12 '13 at 8:08

116

Show code snippet

If ID's are generated more than 1 millisecond apart, they are 100% unique.

If two ID's are generated at shorter intervals, and assuming that the random method is truly random, this would generate ID's that are 99.99999999999999% likely to be globally unique (collision in 1 of 10^15)

You can increase this number by adding more digits, but to generate 100% unique ID's you will need to use a global counter.

if you need RFC compatibility, this formatting will pass as a valid version 4 GUID:

```
let u = Date.now().toString(16) + Math.random().toString(16) + '0'.repeat(16);
let guid = [u.substr(0,8), u.substr(8,4), '4000-8' + u.substr(13,3),
u.substr(16,12)].join('-');
```

Show code snippet

Edit: The above code follow the intention, but not the letter of the RFC. Among other discrepancies it's a few random digits short. (Add more random digits if you need it) The upside is that this it's really fast, compared to 100% compliant code. You can test validity of your GUID here

edited Jul 15 at 21:16

community wiki
15 revs
Simon Rigét

3 This is not UUID though? – Marco Kerwitz Dec 26 '17 at 23:28

No. UUID/GUID's is a 122 bit (+ six reserved bits) number. it might guarantee uniqueness through a global counter service, but often it relays on time, MAC address and randomness. UUID's are not random! The UID I suggest here is not fully compressed. You could compress it, to a 122 bit integer, add the 6 predefined bits and extra random bits (remove a few timer bits) and you end up with a perfectly formed UUID/GUID, that you would then have to convert to hex. To me that doesn't really add anything other than compliance to the length of the ID. – Simon Rigét Feb 18 '18 at 0:05

4 Relaying on MAC addresses for uniqueness on virtual machines is a bad idea! – Simon Rigét Feb 18 '18 at 0:48

1 I do something like this, but with leading characters and some dashes (e.g `[slug, date, random].join("_")` to create `usr_1dcn27itd_hj6onj6phr` . It makes it so the id also doubles as a "created at" field. – Sophie Read Jun 6 at 19:23

86

compliant GUID.

Ten million executions of this implementation take just 32.5 seconds, which is the fastest I've ever seen in a browser (the only solution without loops/iterations).

The function is as simple as:

```
/**
 * Generates a GUID string.
 * @returns {String} The generated GUID.
 * @example af8a8416-6e18-a307-bd9c-f2c947bbb3aa
 * @author Slavik Meltser (slavik@meltser.info).
 * @link http://slavik.meltser.info/?p=142
 */
function guid() {
    function _p8(s) {
        var p = (Math.random().toString(16)+"000000000").substr(2,8);
        return s ? "-" + p.substr(0,4) + "-" + p.substr(4,4) : p ;
    }
    return _p8() + _p8(true) + _p8(true) + _p8();
}
```

To test the performance, you can run this code:

```
console.time('t');
for (var i = 0; i < 10000000; i++) {
    guid();
};
console.timeEnd('t');
```

I'm sure most of you will understand what I did there, but maybe there is at least one person that will need an explanation:

The algorithm:

- The `Math.random()` function returns a decimal number between 0 and 1 with 16 digits after the decimal fraction point (for example `0.4363923368509859`).
- Then we take this number and convert it to a string with base 16 (from the example above we'll get `0.6fb7687f`).

- Sometimes the `Math.random()` function will return shorter number (for example `0.4363`), due to zeros at the end (from the example above, actually the number is `0.4363000000000000`). That's why I'm appending to this string `"000000000"` (a string with nine zeros) and then cutting it off with `substr()` function to make it nine characters exactly (filling zeros to the right).
- The reason for adding exactly nine zeros is because of the worse case scenario, which is when the `Math.random()` function will return exactly 0 or 1 (probability of 1/10^16 for each one of them). That's why we needed to add nine zeros to it (`"0"+"000000000"` or `"1"+"000000000"`), and then cutting it off from the second index (3rd character) with a length of eight characters. For the rest of the cases, the addition of zeros will not harm the result because it is cutting it off anyway.
  `Math.random().toString(16)+"000000000").substr(2,8)`.

The assembly:

- The GUID is in the following format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.
- I divided the GUID into 4 pieces, each piece divided into 2 types (or formats): `xxxxxxxx` and `-xxxx-xxxx`.
- Now I'm building the GUID using these 2 types to assemble the GUID with call 4 pieces, as follows: `xxxxxxxx -xxxx-xxxx -xxxx-xxxx xxxxxxxx`.
- To differ between these two types, I added a flag parameter to a pair creator function `_p8(s)`, the `s` parameter tells the function whether to add dashes or not.
- Eventually we build the GUID with the following chaining: `_p8() + _p8(true) + _p8(true) + _p8()`, and return it.

Link to this post on my blog

Enjoy! :-)

edited Jul 15 '18 at 19:10

community wiki
12 revs, 3 users 95%
Slavik Meltser

13 This implementation is incorrect. Certain characters of the GUID require special treatment (e.g. the 13th digit needs to be the number 4). – JLRishe Nov 12 '13 at 8:12

1 @JLRishe, you are right, it doesn't follow the RFC4122 standards. But it's still a random string that looks like GUID. Cheers :-) – Slavik Meltser Nov 24 '13 at 22:33

3 Nice work, but classic optimization techniques make it 6X faster (on my browser) - see my answer – Jeff Ward Feb 25 '14 at 19:23

62

```
generateGUID = (typeof(window.crypto) != 'undefined' &&
                typeof(window.crypto.getRandomValues) != 'undefined') ?
    function() {
        // If we have a cryptographically secure PRNG, use that
        // https://stackoverflow.com/questions/6906916/collisions-when-generating-uuids-
in-javascript
        var buf = new Uint16Array(8);
        window.crypto.getRandomValues(buf);
        var S4 = function(num) {
            var ret = num.toString(16);
            while(ret.length < 4){
                ret = "0"+ret;
            }
            return ret;
        };
        return (S4(buf[0])+S4(buf[1])+"-"+S4(buf[2])+"-"+S4(buf[3])+"-"+S4(buf[4])+"-
"+S4(buf[5])+S4(buf[6])+S4(buf[7]));
    }

    :

    function() {
        // Otherwise, just use Math.random
        // https://stackoverflow.com/questions/105034/how-to-create-a-guid-uuid-in-
javascript/2117523#2117523
        return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
            var r = Math.random()*16|0, v = c == 'x' ? r : (r&0x3|0x8);
            return v.toString(16);
        });
    };
```

On jsbin if you want to test it.

edited May 23 '17 at 11:47

community wiki
3 revs
ripper234

2 I believe that in IE it's actually window.msCrypto instead of window.crypto. Might be nice to check for both. See msdn.microsoft.com/en-us/library/ie/dn265046(v=vs.85).aspx – herbrandson Apr 17 '14 at 20:44

57

Here is a totally non-compliant but very performant implementation to generate an ASCII-safe GUID-like unique identifier.

```
function generateQuickGuid() {
    return Math.random().toString(36).substring(2, 15) +
        Math.random().toString(36).substring(2, 15);
}
```

Generates 26 [a-z0-9] characters, yielding a UID that is both shorter and more unique than RFC compliant GUIDs. Dashes can be trivially added if human-readability matters.

Here are usage examples and timings for this function and several of this question's other answers. The timing was performed under Chrome m25, 10 million iterations each.

```
>>> generateQuickGuid()
"nvcjf1hs7tf8yyk4lmlijqkuo9"
"yq6gipxqta4kui8z05tgh9qeel"
"36dh5sec7zdj90sk2rx7pjswi2"
runtime: 32.5s

>>> GUID() // John Millikin
"7a342ca2-e79f-528e-6302-8f901b0b6888"
runtime: 57.8s

>>> regexGuid() // broofa
"396e0c46-09e4-4b19-97db-bd423774a4b3"
runtime: 91.2s

>>> createUUID() // Kevin Hakanson
"403aa1ab-9f70-44ec-bc08-5d5ac56bd8a5"
runtime: 65.9s

>>> UUIDv4() // Jed Schmidt
"f4d7d31f-fa83-431a-b30c-3e6cc37cc6ee"
runtime: 282.4s

>>> Math.uuid() // broofa
"5BD52F55-E68F-40FC-93C2-90EE069CE545"
runtime: 225.8s

>>> Math.uuidFast() // broofa
```

```
runtime: 229.0s

>>> bitwiseGUID() // jablko
"baeaa2f-7587-4ff1-af23-eeab3e92"
runtime: 79.6s

>>>> betterWayGUID() // Andrea Turri
"383585b0-9753-498d-99c3-416582e9662c"
runtime: 60.0s

>>>> UUID() // John Fowler
"855f997b-4369-4cdb-b7c9-7142ceaf39e8"
runtime: 62.2s
```

Here is the timing code.

```
var r;
console.time('t');
for (var i = 0; i < 10000000; i++) {
    r = FuncToTest();
};
console.timeEnd('t');
```

edited Jan 21 '13 at 21:52

community wiki
joelpt

56

Here's a solution dated Oct. 9, 2011 from a comment by user *jed* at https://gist.github.com/982883:

```
UUIDv4 = function b(a){return a?(a^Math.random()*16>>a/4).toString(16):
([1e7]+-1e3+-4e3+-8e3+-1e11).replace(/[018]/g,b)}
```

This accomplishes the same goal as the current highest-rated answer, but in 50+ fewer bytes by exploiting coercion, recursion, and exponential notation. For those curious how it works, here's the annotated form of an older version of the function:

```
  return a // if the placeholder was passed, return
    ? ( // a random number from 0 to 15
      a ^ // unless b is 8,
      Math.random() // in which case
      * 16 // a random number from
      >> a/4 // 8 to 11
      ).toString(16) // in hexadecimal
    : ( // or otherwise a concatenated string:
      [1e7] + // 10000000 +
      -1e3 + // -1000 +
      -4e3 + // -4000 +
      -8e3 + // -80000000 +
      -1e11 // -100000000000,
      ).replace( // replacing
        /[018]/g, // zeroes, ones, and eights with
        b // random hex digits
      )
}
```

edited May 23 '17 at 10:31

community wiki
9 revs, 5 users 63%
Jed Schmidt

1 In TypeScript use this: `export const UUID = function b (a: number): string { return a ? (a^Math.random()*16>>a/4).toString(16) : (''+ [1e7]+-1e3+-4e3+-8e3+-1e11).replace(/[018]/g, b) };` – RyanNerd Aug 7 '18 at 22:45

38

From [sagi shkedy's technical blog]():

```
function generateGuid() {
  var result, i, j;
  result = '';
  for(j=0; j<32; j++) {
    if( j == 8 || j == 12 || j == 16 || j == 20)
      result = result + '-';
    i = Math.floor(Math.random()*16).toString(16).toUpperCase();
    result = result + i;
  }
```

Edit : I thought it was worth pointing out that no GUID generator can guarantee unique keys (check the wikipedia article). There is always a chance of collisions. A GUID simply offers a large enough universe of keys to reduce the change of collisions to almost nil.

edited Nov 26 '18 at 6:57

community wiki
4 revs, 3 users 89%
Prestaul

8 Note that this isn't a GUID in the technical sense, because it does nothing to guarantee uniqueness. That may or may not matter depending on your application. – Stephen Deken Sep 19 '08 at 20:07

Ditto to Stephen's response. If you need uniqueness, define it server side where hopefully to can get to a proper algorithm! – Ray Hayes Sep 19 '08 at 20:08

7 No GUID is guaranteed to be unique... The universe of created keys is simply large enough to make collisions nearly impossible. – Prestaul Sep 19 '08 at 20:13

2 A quick note about performance. This solution creates 36 strings total to get a single result. If performance is critical, consider creating an array and joining as recommended by: tinyurl.com/y37xtx Further research indicates it may not matter, so YMMV: tinyurl.com/3l7945 – Brandon DuRette Sep 22 '08 at 18:14

1 Regarding uniqueness, it's worth noting that version 1,3, and 5 UUIDs are deterministic in ways version 4 isn't. If the inputs to these uuid generators - node id in v1, namespace and name in v3 and v5 - are unique (as they're supposed to be), then the resulting UUIDs be unique. In theory, anyway. – broofa Jun 29 '17 at 13:26

35

You can use node-uuid (https://github.com/kelektiv/node-uuid)

Simple, fast generation of RFC4122 UUIDS.

Features:

- Generate RFC4122 version 1 or version 4 UUIDs
- Runs in node.js and browsers.
- Cryptographically strong random # generation on supporting platforms.
- Small footprint (Want something smaller? Check this out!)

Or Using uuid via browser:

Download Raw File (uuid v1): https://raw.githubusercontent.com/kelektiv/node-uuid/master/v1.js Download Raw File (uuid v4): https://raw.githubusercontent.com/kelektiv/node-uuid/master/v4.js

Want even smaller? Check this out: https://gist.github.com/jed/982883

Usage:

```
// Generate a v1 UUID (time-based)
const uuidV1 = require('uuid/v1');
uuidV1(); // -> '6c84fb90-12c4-11e1-840d-7b25c5ee775a'

// Generate a v4 UUID (random)
const uuidV4 = require('uuid/v4');
uuidV4(); // -> '110ec58a-a0f2-4ac4-8393-c866d813b8d1'

// Generate a v5 UUID (namespace)
const uuidV5 = require('uuid/v5');

// ... using predefined DNS namespace (for domain names)
uuidV5('hello.example.com', v5.DNS)); // -> 'fdda765f-fc57-5604-a269-52a7df8164ec'

// ... using predefined URL namespace (for, well, URLs)
uuidV5('http://example.com/hello', v5.URL); // -> '3bbcee75-cecc-5b56-8031-b6641c1ed1f1'

// ... using a custom namespace
const MY_NAMESPACE = '(previously generated unique uuid string)';
uuidV5('hello', MY_NAMESPACE); // -> '90123e1c-7512-523e-bb28-76fab9f2f73d'
```

ES6:

6 revs, 2 users 99%
Kyros Koh

The Raw File is dead. – Edward Olamisan Apr 13 '17 at 3:50

1 @Edward Olamisan Updated new source to replace the legacy version. – Kyros Koh Apr 14 '17 at 21:31

1 ES6 version: `import uuid from 'uuid/v4'; let id = uuid();` – justin.m.chase May 20 '17 at 5:30

@justin.m.chase shouldn't that be const id = uuid – Kermit_ice_tea Feb 22 '18 at 1:07

31

A web service would be useful.

Quick Google found: http://www.hoskinson.net/GuidGenerator/

Can't vouch for this implementation, but SOMEONE must publish a bonafide GUID generator.

With such a web service, you could develop a REST web interface that consumes the GUID web service, and serves it through AJAX to javascript in a browser.

edited Sep 19 '08 at 20:35

community wiki
Sean

10 I made, host and use this one: timjeanes.com/guid. It uses .NET to generate a new GUID and returns it without any additional fluff. It'll also work over JSONP. – teedyay Jun 2 '10 at 20:35

8 A web service to serve a GUID, really? That's almost as strange as writing a web service to serve random numbers -- unless you need really truly random numbers produced by some physical noise source attached to a server, that is. – Pierre Arnaud Sep 9 '14 at 6:21

The only problem with a network service is that the network might be down. – Gustav Dec 26 '17 at 19:09

Link is outdated, service is out of order. – Marecky Mar 4 '18 at 18:16

```
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
        idx++;
        var r = (buf[idx>>3] >> ((idx%8)*4))&15;
        var v = c == 'x' ? r : (r&0x3|0x8);
        return v.toString(16);
    });
};
```

EDIT:

Revisited my project that was using this function and disliked the verbosity. - But needed proper randomness.

A version based on Briguy37's answer and some bitwise operators to extract nibble sized windows from the buffer.

Should adhere to the RFC Type 4 (random) schema, since I had Problems last time parsing non-compliant uuids with Java's UUID.

edited Sep 11 '15 at 22:32

community wiki
5 revs, 2 users 98%
sleeplessnerd

On jsbin: jsbin.com/uqives/2 It's worth stating more explicitly that this doesn't work on IE. Didn't work for Firefox also in my test. – ripper234 Dec 12 '11 at 10:04

Thanks for trying it. I whipped it up for an chrome Extension. I did not check the facts I read on the Internt(tm) about firefox supporting it. It is also worth mentioning that is does not comply with the RFC, the random (Version 4) UUID should have two places with fixed values: en.wikipedia.org/wiki/… – sleeplessnerd Dec 12 '11 at 17:03

29

Simple JavaScript module as a combination of best answers in this thread.

```
var crypto = window.crypto || window.msCrypto || null; // IE11 fix

var Guid = Guid || (function() {

  var EMPTY = '00000000-0000-0000-0000-000000000000';
```

```
    var hexadecimalResult = number.toString(16);
    return _padLeft(hexadecimalResult, 4, '0');
  };

  var _cryptoGuid = function() {
    var buffer = new window.Uint16Array(8);
    window.crypto.getRandomValues(buffer);
    return [_s4(buffer[0]) + _s4(buffer[1]), _s4(buffer[2]), _s4(buffer[3]),
_s4(buffer[4]), _s4(buffer[5]) + _s4(buffer[6]) + _s4(buffer[7])].join('-');
  };

  var _guid = function() {
    var currentDateMilliseconds = new Date().getTime();
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(currentChar)
{
      var randomChar = (currentDateMilliseconds + Math.random() * 16) % 16 | 0;
      currentDateMilliseconds = Math.floor(currentDateMilliseconds / 16);
      return (currentChar === 'x' ? randomChar : (randomChar & 0x7 | 0x8)).toString(16);
    });
  };

  var create = function() {
    var hasCrypto = crypto != 'undefined' && crypto !== null,
      hasRandomValues = typeof(window.crypto.getRandomValues) != 'undefined';
    return (hasCrypto && hasRandomValues) ? _cryptoGuid() : _guid();
  };

  return {
    newGuid: create,
    empty: EMPTY
  };
})();

// DEMO: Create and show GUID
console.log(Guid.newGuid());
```

Run code snippet Expand snippet

Usage:

> **Guid.empty**
>
> "00000000-0000-0000-0000-000000000000"

edited Jun 29 '17 at 13:29

community wiki
4 revs, 3 users 72%
kayz1

1 What is bothering about *all* answers is that it seems *ok* for JavaScript to store the `GUID` as a `string`. Your answer at least tackles the *much* more efficient storage using a `Uint16Array`. The `toString` function should be using the binary representation in an JavaScript `object` – Sebastian May 4 '14 at 11:03

2 Shouldn't `_cryptoGuid` include a '4' at the start of the 3rd section like `_guid` does? – row1 Feb 1 '15 at 21:05

Good answer! I converted the code into a snippet and fixed IE11 compatibility issue, according to **this question.** – Matt May 18 '17 at 11:56

This UUIDs produced by this code are either weak-but-RFC-compliant (_guid), or strong-but-not-RFC-compliant (_cryptoGuid). The former uses Math.random(), which is now known to be a poor RNG. The latter is failing to set the version and variant fields. – broofa Jun 29 '17 at 13:37

@broofa - What would you suggest to make it strong **and** RFC-compliant? And why is _cryptoGuid not RFC-compliant? – Matt Mar 15 '18 at 9:57

25

From good ol' wikipedia there's a link to a javascript implementation of UUID.

It looks fairly elegant, and could perhaps be improved by salting with a hash of the client's IP address. This hash could perhaps be inserted into the html document server-side for use by the client-side javascript.

UPDATE : The original site has had a shuffle, here is the updated version

edited Feb 16 '11 at 19:25

community wiki
Dan

2 The link is dead. Can you provide an alternative? – Will Jan 12 '11 at 14:04

24

Well, this has a bunch of answers already, but unfortunately there's not a "true" random in the bunch. The version below is an adaptation of broofa's answer, but updated to include a "true" random function that uses crypto libraries where available, and the Alea() function as a fallback.

```
Math.log2 = Math.log2 || function(n){ return Math.log(n) / Math.log(2); }
Math.trueRandom = (function() {
var crypt = window.crypto || window.msCrypto;

if (crypt && crypt.getRandomValues) {
    // if we have a crypto library, use it
    var random = function(min, max) {
        var rval = 0;
        var range = max - min;
        if (range < 2) {
            return min;
        }

        var bits_needed = Math.ceil(Math.log2(range));
        if (bits_needed > 53) {
          throw new Exception("We cannot generate numbers larger than 53 bits.");
        }
        var bytes_needed = Math.ceil(bits_needed / 8);
        var mask = Math.pow(2, bits_needed) - 1;
        // 7776 -> (2^13 = 8192) -1 == 8191 or 0x00001111 11111111

        // Create byte array and fill with N random numbers
        var byteArray = new Uint8Array(bytes_needed);
        crypt.getRandomValues(byteArray);

        var p = (bytes_needed - 1) * 8;
        for(var i = 0; i < bytes_needed; i++ ) {
            rval += byteArray[i] * Math.pow(2, p);
            p -= 8;
        }

        // Use & to apply the mask and reduce the number of recursive lookups
        rval = rval & mask;

        if (rval >= range) {
            // Integer out of acceptable range
            return random(min, max);
```

```
            return r;
        };
    } else {
        // From http://baagoe.com/en/RandomMusings/javascript/
        // Johannes BaagÃ¸e <baagoe@baagoe.com>, 2010
        function Mash() {
            var n = 0xefc8249d;

            var mash = function(data) {
                data = data.toString();
                for (var i = 0; i < data.length; i++) {
                    n += data.charCodeAt(i);
                    var h = 0.02519603282416938 * n;
                    n = h >>> 0;
                    h -= n;
                    h *= n;
                    n = h >>> 0;
                    h -= n;
                    n += h * 0x100000000; // 2^32
                }
                return (n >>> 0) * 2.3283064365386963e-10; // 2^-32
            };

            mash.version = 'Mash 0.9';
            return mash;
        }

        // From http://baagoe.com/en/RandomMusings/javascript/
        function Alea() {
            return (function(args) {
                // Johannes BaagÃ¸e <baagoe@baagoe.com>, 2010
                var s0 = 0;
                var s1 = 0;
                var s2 = 0;
                var c = 1;

                if (args.length == 0) {
                    args = [+new Date()];
                }
                var mash = Mash();
                s0 = mash(' ');
                s1 = mash(' ');
                s2 = mash(' ');
```

```
                    s1 -= mash(args[i]);
                    if (s1 < 0) {
                        s1 += 1;
                    }
                    s2 -= mash(args[i]);
                    if (s2 < 0) {
                        s2 += 1;
                    }
                }
                mash = null;

                var random = function() {
                    var t = 2091639 * s0 + c * 2.3283064365386963e-10; // 2^-32
                    s0 = s1;
                    s1 = s2;
                    return s2 = t - (c = t | 0);
                };
                random.uint32 = function() {
                    return random() * 0x100000000; // 2^32
                };
                random.fract53 = function() {
                    return random() +
                        (random() * 0x200000 | 0) * 1.1102230246251565e-16; // 2^-53
                };
                random.version = 'Alea 0.9';
                random.args = args;
                return random;

            }(Array.prototype.slice.call(arguments)));
        };
        return Alea();
    }
}());

Math.guid = function() {
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c)    {
        var r = Math.trueRandom() * 16 | 0,
            v = c == 'x' ? r : (r & 0x3 | 0x8);
        return v.toString(16);
    });
};
```

2 You need to comment out the line `rval = rval & mask` since it truncates the result to 32 bits. Bit ops are only 32 bits in JS. Moreover, if you apply it only at the end then the 3 low bits are already lost, when generating 53-bit value, since the first loop creates a 56-bit value (255 * 2^48). Instead, you need to use this code: ` var byte = byteArray[i]; if (p + 8 > bits_needed) byte &= (255 >> p + 8 - bits_needed); rval += byte * Math.pow(2, p);` You can see the test here: jsfiddle.net/xto6969y/1 – Roland Pihlakas May 2 '15 at 8:57

3 Also you may consider using var crypto = window.crypto || window.msCrypto see msdn.microsoft.com/library/dn265046(v=vs.85).aspx – Roland Pihlakas May 2 '15 at 9:00

1 Updated on both counts. – jvenema Jun 2 '15 at 14:10

I`ve use your script and got "Uncaught RangeError: Maximum call stack size exceeded(…)" on var byteArray = new Uint8Array(bytes_needed); What could be a problem? – AlexBerd Apr 6 '16 at 8:35

1 Bad edits in the past, should be good now. – jvenema Apr 11 '16 at 13:51

23

This create version 4 UUID (created from pseudo random numbers) :

```
function uuid()
{
   var chars = '0123456789abcdef'.split('');

   var uuid = [], rnd = Math.random, r;
   uuid[8] = uuid[13] = uuid[18] = uuid[23] = '-';
   uuid[14] = '4'; // version 4

   for (var i = 0; i < 36; i++)
   {
      if (!uuid[i])
      {
         r = 0 | rnd()*16;

         uuid[i] = chars[(i == 19) ? (r & 0x3) | 0x8 : r & 0xf];
      }
   }

   return uuid.join('');
}
```

answered Aug 24 '09 at 16:12

community wiki
Mathieu Pagé

JavaScript project on GitHub - https://github.com/LiosK/UUID.js

23

UUID.js The RFC-compliant UUID generator for JavaScript.

See RFC 4122 http://www.ietf.org/rfc/rfc4122.txt.

Features Generates RFC 4122 compliant UUIDs.

Version 4 UUIDs (UUIDs from random numbers) and version 1 UUIDs (time-based UUIDs) are available.

UUID object allows a variety of access to the UUID including access to the UUID fields.

Low timestamp resolution of JavaScript is compensated by random numbers.

answered Jul 2 '12 at 21:00

community wiki
Wojciech Bednarski

16

```
// RFC 4122
//
// A UUID is 128 bits long
//
// String representation is five fields of 4, 2, 2, 2, and 6 bytes.
// Fields represented as lowercase, zero-filled, hexadecimal strings, and
// are separated by dash characters
//
// A version 4 UUID is generated by setting all but six bits to randomly
// chosen values
```

```
    // 4.1.3
    (Math.random() * .0625 /* 0x.1 */ + .25 /* 0x.4 */).toString(16).slice(2, 6),

    // Set the two most significant bits (bits 6 and 7) of the
    // clock_seq_hi_and_reserved to zero and one, respectively
    (Math.random() * .25 /* 0x.4 */ + .5 /* 0x.8 */).toString(16).slice(2, 6),

    Math.random().toString(16).slice(2, 14)].join('-');
```

answered Jul 14 '10 at 23:30 — community wiki jablko

1 I like this approach, but beware that it does not work properly in Chrome. The ".slice(2, 14)" portion only returns 8 characters, not 12. – jalbert Sep 14 '11 at 20:37

13

Adjusted my own UUID/GUID generator with some extras here.

I'm using the following Kybos random number generator to be a bit more cryptographically sound.

Below is my script with the Mash and Kybos methods from baagoe.com excluded.

```
//UUID/Guid Generator
// use: UUID.create() or UUID.createSequential()
// convenience:  UUID.empty, UUID.tryParse(string)
(function(w){
  // From http://baagoe.com/en/RandomMusings/javascript/
  // Johannes BaagÃ¸e <baagoe@baagoe.com>, 2010
  //function Mash() {...};

  // From http://baagoe.com/en/RandomMusings/javascript/
  //function Kybos() {...};

  var rnd = Kybos();

  //UUID/GUID Implementation from http://frugalcoder.us/post/2012/01/13/javascript-guid-
```

```
      if ((/[a-f0-9]{8}\-[a-f0-9]{4}\-[a-f0-9]{4}\-[a-f0-9]{4}\-[a-f0-9]
{12}/).test(ret))
        return ret;
      else
        throw new Error("Unable to parse UUID");
    }
    ,"createSequential": function() {
      var ret = new Date().valueOf().toString(16).replace("-","")
      for (;ret.length < 12; ret = "0" + ret);
      ret = ret.substr(ret.length-12,12); //only least significant part
      for (;ret.length < 32;ret += Math.floor(rnd() * 0xffffffff).toString(16));
      return [ret.substr(0,8), ret.substr(8,4), "4" + ret.substr(12,3), "89AB"
[Math.floor(Math.random()*4)] + ret.substr(16,3),  ret.substr(20,12)].join("-");
    }
    ,"create": function() {
      var ret = "";
      for (;ret.length < 32;ret += Math.floor(rnd() * 0xffffffff).toString(16));
      return [ret.substr(0,8), ret.substr(8,4), "4" + ret.substr(12,3), "89AB"
[Math.floor(Math.random()*4)] + ret.substr(16,3),  ret.substr(20,12)].join("-");
    }
    ,"random": function() {
      return rnd();
    }
    ,"tryParse": function(input) {
      try {
        return UUID.parse(input);
      } catch(ex) {
        return UUID.empty;
      }
    }
  };
  UUID["new"] = UUID.create;

  w.UUID = w.Guid = UUID;
}(window || this));
```

answered Jan 13 '12 at 21:59

community wiki
Tracker1

```
){
  for(               // loop :)
      b=a='';        // b - result , a - numeric variable
      a++<36;        //
      b+=a*51&52  // if "a" is not 9 or 14 or 19 or 24
                  ?  //  return a random number or 4
         (
           a^15      // if "a" is not 15
              ?      // genetate a random number from 0 to 15
           8^Math.random()*
           (a^20?16:4)  // unless "a" is 20, in which case a random number from 8 to 11
              :
           4            //  otherwise 4
           ).toString(16)
                  :
         '-'            //  in other cases (if "a" is 9,14,19,24) insert "-"
      );
  return b
 }
```

Minimized:

```
function(a,b){for(b=a='';a++<36;b+=a*51&52?(a^15?8^Math.random()*(a^20?
16:4):4).toString(16):'-');return b}
```

answered May 23 '12 at 18:42 community wiki Andrea Turri

12

I wanted to understand broofa's answer, so I expanded it and added comments:

```
var uuid = function () {
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(
        /[xy]/g,
        function (match) {
            /*
             * Create a random nibble. The two clever bits of this code:
```

```
                *
                * Math.random * 16
                *
                * creates a random floating point number
                * between 0 (inclusive) and 16 (exclusive) and
                *
                * | 0
                *
                * truncates the floating point number into an integer.
                */
                var randomNibble = Math.random() * 16 | 0;

                /*
                * Resolves the variant field. If the variant field (delineated
                * as y in the initial string) is matched, the nibble must
                * match the mask (where x is a do-not-care bit):
                *
                * 10xx
                *
                * This is achieved by performing the following operations in
                * sequence (where x is an intermediate result):
                *
                * - x & 0x3, which is equivalent to x % 3
                * - x | 0x8, which is equivalent to x + 8
                *
                * This results in a nibble between 8 inclusive and 11 exclusive,
                * (or 1000 and 1011 in binary), all of which satisfy the variant
                * field mask above.
                */
                var nibble = (match == 'y') ?
                    (randomNibble & 0x3 | 0x8) :
                    randomNibble;

                /*
                * Ensure the nibble integer is encoded as base 16 (hexadecimal).
                */
                return nibble.toString(16);
            }
        );
};
```

edited Feb 20 '17 at 14:46 community wiki

ES6 sample

12

```
const guid=()=> {
  const s4=()=> Math.floor((1 + Math.random()) * 0x10000).toString(16).substring(1);
  return `${s4() + s4()}-${s4()}-${s4()}-${s4()}-${s4() + s4() + s4()}`;
}
```

answered Jul 9 '17 at 13:01

community wiki
Behnam Mohammadi

11

It's just a simple AJAX call...

If anyone is still interested, here's my solution.

On the server side:

```
[WebMethod()]
public static string GenerateGuid()
{
    return Guid.NewGuid().ToString();
}
```

On the client side:

```
var myNewGuid = null;
PageMethods.GenerateGuid(
    function(result, userContext, methodName)
    {
        myNewGuid = result;
    },
    function()
    {
        alert("WebService call failed.");
```

alekop

Your method is the only correct way, but the problem with it is that it's asynchronous, so you can't really use that. Besides, try doing that a few 100 to 1000 times, and you will crash IE (not Chrome and Firefox, though). Synchronous calls would we needed: Use JQuery, not MS-PageMethod JavaScript ! – Stefan Steiger Mar 30 '10 at 11:46

2 You're right, called asynchronously this is not very useful. Ironically my original code does use jQuery to invoke this method synchronously. Here's an example: $.ajax({ async: false, type: 'POST', url: 'MyPage.aspx/GenerateGuid', contentType: 'application/json; charset=utf-8', data: '{}', success: function(data) { // data contains your new GUID }, failure: function(msg) { alert(msg); } }); – alekop Apr 5 '10 at 23:26

10 Why do you need to make AJAX call if you are using ASP.NET? Just do <%= Guid.NewGuid().ToString() %> in aspx. – nikib3ro Jan 26 '12 at 1:02

2 @kape123: That's fine, if you only want one GUID. The Web service allows you to generate multiple GUIDs without reloading the page. – alekop Jul 23 '13 at 1:20

11

For those wanting an rfc4122 version 4 compliant solution with speed considerations (few calls to Math.random()):

```
function UUID() {
    var nbr, randStr = "";
    do {
        randStr += (nbr = Math.random()).toString(16).substr(2);
    } while (randStr.length < 30);
    return [
        randStr.substr(0, 8), "-",
        randStr.substr(8, 4), "-4",
        randStr.substr(12, 3), "-",
        ((nbr*4|0)+8).toString(16), // [89ab]
        randStr.substr(15, 3), "-",
        randStr.substr(18, 12)
        ].join("");
}
```

The above function should have a decent balance between speed and randomness.

answered Nov 16 '12 at 19:41 community wiki John Fowler

11

like and is easy to hack:

uid-139410573297741

```
var getUniqueId = function (prefix) {
            var d = new Date().getTime();
            d += (parseInt(Math.random() * 100)).toString();
            if (undefined === prefix) {
                prefix = 'uid-';
            }
            d = prefix + d;
            return d;
        };
```

answered Mar 6 '14 at 11:34

community wiki
ling

10

I know, it is an old question. Just for completeness, if your environment is SharePoint, there is a utility function called `SP.Guid.newGuid` (msdn link) which creates a new guid. This function is inside the sp.init.js file. If you rewrite this function (to remove some other dependencies from other private functions), it looks like this:

```
var newGuid = function () {
    var result = '';
    var hexcodes = "0123456789abcdef".split("");

    for (var index = 0; index < 32; index++) {
        var value = Math.floor(Math.random() * 16);

        switch (index) {
        case 8:
            result += '-';
            break;
        case 12:
            value = 4;
            result += '-';
```

```
                result += '-';
                break;
            }
            result += hexcodes[value];
        }
        return result;
    };
```

answered Jun 12 '13 at 16:00 community wiki Anatoly Mironov

10

There is a jQuery plugin that handles Guid's nicely @ http://plugins.jquery.com/project/GUID_Helper

```
jQuery.Guid.Value()
```

Returns value of internal Guid. If no guid has been specified, returns a new one (value is then stored internally).

```
jQuery.Guid.New()
```

Returns a new Guid and sets it's value internally.

```
jQuery.Guid.Empty()
```

Returns an empty Guid 00000000-0000-0000-0000-000000000000.

```
jQuery.Guid.IsEmpty()
```

Returns boolean. True valid guid, false if not.

```
jQuery.Guid.Set()
```

Retrns Guid. Sets Guid to user specified Guid, if invalid, returns an empty guid.

edited Oct 9 '13 at 4:54

community wiki
Levitikon

1 The link to the plugin is broken – Matt May 18 '17 at 8:08

9

Simple code that uses `crypto.getRandomValues(a)` on supported browsers (IE11+, iOS7+, FF21+, Chrome, Android Chrome). Avoids using `Math.random()` because that can cause collisions (for example 20 collisions for 4000 generated uuids in a real situation by Muxa).

```
function uuid() {
    function randomDigit() {
        if (crypto && crypto.getRandomValues) {
            var rands = new Uint8Array(1);
            crypto.getRandomValues(rands);
            return (rands[0] % 16).toString(16);
        } else {
            return ((Math.random() * 16) | 0).toString(16);
        }
    }
    var crypto = window.crypto || window.msCrypto;
    return 'xxxxxxxx-xxxx-4xxx-8xxx-xxxxxxxxxxxx'.replace(/x/g, randomDigit);
}
```

Notes:

- Optimised for code readability not speed, so suitable for say a few hundred uuid's per second. Generates about 10000 uuid() per second in Chromium on my laptop using http://jsbin.com/fuwigo/1 to measure performance

2 revs
robocat

1 jsfiddle.net/of3v5zko Jsfiddle that Uses crypto API to generate a UUID, compliant with RFC4122 version 4 Original source github.com/Chalarangelo/30-seconds-of-code#uuid-generator – Ananda Dec 14 '17 at 9:12

@ananda Thanks. However that the code is not backwards compatible with existing browsers. Also although that code saves a few bytes I think it is less readable (I like my code to be clear) and it is quite possible that after uglifier and gzipping your version saves nothing (a run of `x` characters and common identifiers compress very well). – robocat 8 mins ago – robocat Mar 11 '18 at 22:48

1 2 next

protected by NullPointer Jun 10 '13 at 5:10

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?