How can I generate random alphanumeric strings? [closed]

Ask Question

How can I generate random 8 character alphanumeric strings in C#?

c# .net random

edited Nov 7 '18 at 6:36

Uwe Keim
27.7k 32 135 216

asked Aug 27 '09 at 23:07

KingNestor
25k 43 108 145

closed as too broad by meagar ♦ Feb 25 '16 at 16:24

Please edit the question to limit it to a specific problem with enough detail to identify an adequate answer. Avoid asking multiple distinct questions at once. See the How to Ask page for help clarifying this question.

If this question can be reworded to fit the rules in the help center, please edit the question.

2 What restrictions if any do you have on the character set? Just English language characters and 0-9? Mixed case? – Eric J. Aug 27 '09 at 23:15

8 Or maybe to stackoverflow.com/questions/730268/… or stackoverflow.com/questions/1122483/c-random-string-generator –

Jonas Elfström Aug 27 '09 at 23:15

9 Note that you should NOT use any method based on the `Random` class to generate passwords. The seeding of `Random` has very low entropy, so it's not really secure. Use a cryptographic PRNG for passwords. – CodesInChaos Mar 9 '11 at 18:47

10 Something with this many upvotes and this many quality answers doesn't deserved to be marked as closed. I vote that it be reopened. – John Coleman Oct 16 '17 at 12:52

2 @JohnColeman Lol, welcome to SO. – Ash Apr 16 '18 at 2:16

30 Answers

1455

I heard LINQ is the new black, so here's my attempt using LINQ:

```
private static Random random = new Random();
public static string RandomString(int length)
{
    const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    return new string(Enumerable.Repeat(chars, length)
      .Select(s => s[random.Next(s.Length)]).ToArray());
}
```

*(Note: The use of the `Random` class makes this **unsuitable for anything security related**, such as creating passwords or tokens. Use the `RNGCryptoServiceProvider` class if you need a strong random number generator.)*

edited Mar 1 at 15:18

Liam
16.5k 16 78 131

answered Aug 27 '09 at 23:13

dtb
173k 27 347 404

21 @Alex: I've run a few quick tests and it seems to scale pretty much linearly when generating longer strings (so long as there's actually enough memory available). Having said that, Dan Rigby's answer was almost twice as fast as this one in every test. – LukeH Aug 28 '09 at 0:33

4 Well. If your criteria is that it uses linq and that it has a lousy code narrative then it's definitely the bee's knees. Both the code narrative and actual path of execution is rather inefficient and indirect. Don't get me wrong, I'm a huge code hipster (i love python), but this is pretty much a rube goldberg machine. – eremzeit Sep 29 '11 at 15:41

5 While this technically answers the question, it's output is very misleading. Generating 8 random characters sounds like there can be *very* many results, whereas this at best produces 2 billion different results. And in practice even fewer. You should also add a BIG FAT warning to not use this for anything security related stuff. – CodesInChaos Mar 17 '12 at 20:50

31 @xaisoft: Lowercase letters are left as an exercise for the reader. – dtb Sep 30 '13 at 14:29

10 The following line is more memory (and thus time) efficient than the given one `return new string(Enumerable.Range(1, length).Select(_ => chars[random.Next(chars.Length)]).ToArray());` – Tyson Williams Nov 11 '16 at 18:06

323

```
var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012
var stringChars = new char[8];
var random = new Random();

for (int i = 0; i < stringChars.Length; i++)
{
    stringChars[i] = chars[random.Next(chars.Length)];
}

var finalString = new String(stringChars);
```

Not as elegant as the Linq solution. (-:

*(Note: The use of the `Random` class makes this **unsuitable for anything security related**, such as creating passwords or tokens. Use the `RNGCryptoServiceProvider` class if you need a strong random number generator.)*

edited Mar 1 at 15:18

Liam
16.5k 16 78 131

answered Aug 27 '09 at 23:21

Dan Rigby
11.3k 4 31 56

3 @Alex: This isn't the absolute fastest answer, but it is the fastest "real" answer (ie, of those that allow control over the characters used and the length of the string). – LukeH Aug 28 '09 at 0:56

1 @Alex: Adam Porad's `GetRandomFileName` solution is quicker but doesn't allow any control of the characters used and the max possible length is 11 chars. Douglas's `Guid` solution is lightning-fast but the characters are restricted to A-F0-9 and the max possible length is 32 chars. – LukeH Aug 28 '09 at 1:01

1 @Adam: Yes, you could concat the result of multiple calls to `GetRandomFileName` but then (a) you'd lose your performance advantage, and (b) your code would become more complicated. – LukeH Aug 28 '09 at 9:08

2 @xaisoft Don't use this answer for anything security critical, like passwords. `System.Random` is not suitable for security. – CodesInChaos Oct 21 '13 at 10:14

3 This is definitely better than linq solution. Way easier to read and understand and maintain. – Veysel Özdemir Oct 9 '15 at 15:08

After reviewing the other answers and considering CodeInChaos' comments, along with CodeInChaos still biased (although less)

4

answer, I thought a **final ultimate cut and paste solution** was needed. So while updating my answer I decided to go all out.

For an up to date version of this code, please visit the new Hg repository on Bitbucket: https://bitbucket.org/merarischroeder/secureswiftrandom. I recommend you copy and paste the code from: https://bitbucket.org/merarischroeder/secureswiftrandom/src/6c14b874f34a3f6576b0213379ecdf0ffc7496ea/Code/Alivate.SolidSwiftRandom/SolidSwiftRandom.cs?at=default&fileviewer=file-view-default (make sure you click the Raw button to make it easier to copy and make sure you have the latest version, I think this link goes to a specific version of the code, not the latest).

Updated notes:

1. Relating to some other answers - If you know the length of the output, you don't need a StringBuilder, and when using ToCharArray, this creates and fills the array (you don't need to create an empty array first)
2. Relating to some other answers - You should use NextBytes, rather than getting one at a time for performance
3. Technically you could pin the byte array for faster access.. it's usually worth it when your iterating more than 6-8 times over a byte array. (Not done here)
4. Use of **RNGCryptoServiceProvider for best randomness**
5. Use of **caching of a 1MB buffer of random data** - benchmarking shows cached single bytes access speed is ~1000x faster - taking 9ms over 1MB vs 989ms for uncached.
6. **Optimised rejection of bias zone** within my new class.

End solution to question:

```
static char[] charSet =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".ToC
static int byteSize = 256; //Labelling convenience
```

```
static int biasZone = byteSize - (byteSize % charSet.Length);
public string GenerateRandomString(int Length) //Configurable output
{
    byte[] rBytes = new byte[Length]; //Do as much before and after
    char[] rName = new char[Length];
    SecureFastRandom.GetNextBytesMax(rBytes, biasZone);
    for (var i = 0; i < Length; i++)
    {
        rName[i] = charSet[rBytes[i] % charSet.Length];
    }
    return new string(rName);
}
```

But you need my new (untested) class:

```
/// <summary>
/// My benchmarking showed that for RNGCryptoServiceProvider:
/// 1. There is negligable benefit of sharing RNGCryptoServiceProvid
/// 2. Initial GetBytes takes 2ms, and an initial read of 1MB takes
rise, but still negligable)
/// 2. Cached is ~1000x faster for single byte at a time - taking 9m
for uncached
/// </summary>
class SecureFastRandom
{
    static byte[] byteCache = new byte[1000000]; //My benchmark show
read takes 2ms, and an initial read of this size takes 3ms (starting
    static int lastPosition = 0;
    static int remaining = 0;

    /// <summary>
    /// Static direct uncached access to the RNGCryptoServiceProvide
    /// </summary>
    /// <param name="buffer"></param>
    public static void DirectGetBytes(byte[] buffer)
    {
        using (var r = new RNGCryptoServiceProvider())
        {
            r.GetBytes(buffer);
        }
    }

    /// <summary>
    /// Main expected method to be called by user. Underlying random
RNGCryptoServiceProvider for best performance
    /// </summary>
```

```
        /// <param name="buffer"></param>
        public static void GetBytes(byte[] buffer)
        {
            if (buffer.Length > byteCache.Length)
            {
                DirectGetBytes(buffer);
                return;
            }

            lock (byteCache)
            {
                if (buffer.Length > remaining)
                {
                    DirectGetBytes(byteCache);
                    lastPosition = 0;
                    remaining = byteCache.Length;
                }

                Buffer.BlockCopy(byteCache, lastPosition, buffer, 0, buf
                lastPosition += buffer.Length;
                remaining -= buffer.Length;
            }
        }

        /// <summary>
        /// Return a single byte from the cache of random data.
        /// </summary>
        /// <returns></returns>
        public static byte GetByte()
        {
            lock (byteCache)
            {
                return UnsafeGetByte();
            }
        }

        /// <summary>
        /// Shared with public GetByte and GetBytesWithMax, and not lock
    lock/unlocking in loops. Must be called within lock of byteCache.
        /// </summary>
        /// <returns></returns>
        static byte UnsafeGetByte()
        {
            if (1 > remaining)
            {
                DirectGetBytes(byteCache);
                lastPosition = 0;
```

```
                remaining = byteCache.Length;
            }

            lastPosition++;
            remaining--;
            return byteCache[lastPosition - 1];
        }

        /// <summary>
        /// Rejects bytes which are equal to or greater than max. This is
    ensuring there is no bias when you are modulating with a non power of
        /// </summary>
        /// <param name="buffer"></param>
        /// <param name="max"></param>
        public static void GetBytesWithMax(byte[] buffer, byte max)
        {
            if (buffer.Length > byteCache.Length / 2) //No point caching
            {
                DirectGetBytes(buffer);

                lock (byteCache)
                {
                    UnsafeCheckBytesMax(buffer, max);
                }
            }
            else
            {
                lock (byteCache)
                {
                    if (buffer.Length > remaining) //Recache if not enou
    discarding remaining - too much work to join two blocks
                        DirectGetBytes(byteCache);

                    Buffer.BlockCopy(byteCache, lastPosition, buffer, 0,
                    lastPosition += buffer.Length;
                    remaining -= buffer.Length;

                    UnsafeCheckBytesMax(buffer, max);
                }
            }
        }

        /// <summary>
        /// Checks buffer for bytes equal and above max. Must be called i
    byteCache.
        /// </summary>
        /// <param name="buffer"></param>
```

```
    /// <param name="max"></param>
    static void UnsafeCheckBytesMax(byte[] buffer, byte max)
    {
        for (int i = 0; i < buffer.Length; i++)
        {
            while (buffer[i] >= max)
                buffer[i] = UnsafeGetByte(); //Replace all bytes whi
above max
        }
    }
}
```

For history - my older solution for this answer, used Random object:

```
    private static char[] charSet =
      "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345678

    static rGen = new Random(); //Must share, because the clock seed
(~10ms) resolution, yet lock has only 20-50ns delay.
    static int byteSize = 256; //Labelling convenience
    static int biasZone = byteSize - (byteSize % charSet.Length);
    static bool SlightlyMoreSecurityNeeded = true; //Configuration -
if more security is desired and if charSet.Length is not divisible by
    public string GenerateRandomString(int Length) //Configurable ou
    {
      byte[] rBytes = new byte[Length]; //Do as much before and afte
      char[] rName = new char[Length];
      lock (rGen) //~20-50ns
      {
          rGen.NextBytes(rBytes);

          for (int i = 0; i < Length; i++)
          {
              while (SlightlyMoreSecurityNeeded && rBytes[i] >= biasZ
against 1/5 increased bias of index[0-7] values against others. Note
it == biasZone (that is >=), otherwise there's still a bias on index
                  rBytes[i] = rGen.NextByte();
              rName[i] = charSet[rBytes[i] % charSet.Length];
          }
      }
      return new string(rName);
    }
```

Performance:

1. **SecureFastRandom** - *First single run* = **~9-33ms**. Imperceptible. *Ongoing*: **5ms** (sometimes it goes up to 13ms) over 10,000 iterations, With a single average iteration= **1.5 microseconds.**. Note: Requires generally 2, but occasionally up to 8 cache refreshes - depends on how many single bytes exceed the bias zone
2. **Random** - *First single run* = **~0-1ms**. Imperceptible. *Ongoing*: **5ms** over 10,000 iterations. With a single average iteration= **.5 microseconds.**. About the same speed.

Also check out:

- https://bitbucket.org/merarischroeder/number-range-with-no-bias/src
- https://stackoverflow.com/a/45118325/887092

These links are another approach. Buffering could be added to this new code base, but most important was exploring different approaches to removing bias, and benchmarking the speeds and pros/cons.

edited Nov 30 '18 at 3:25

Todd
10.4k 4 27 43

answered Oct 5 '09 at 6:22

Merari Schroeder
41 1

I found some slight performance enhancements to your method, which seemed the fasted of the bunch -- stackoverflow.com/a/17092645/1037948 – drzaus Jun 13 '13 at 16:46

5 1) Why all those magic constants? You specified the output length *three* times. Just define it as a constant or a parameter. You can use `charSet.Length` instead of `62`. 2) A static `Random` without locking means this code is not threadsafe. 3) reducing 0-255 mod 62 introduces a detectable bias. 4) You can't use `ToString` on a char array, that always

returns `"System.Char[]"` . You need to use `new String(rName)` instead. – CodesInChaos Jun 4 '14 at 10:04

Thanks @CodesInChaos, I never thought of those things back then. Still only using Random class, but this should be better. I couldn't think of any better way to detect and correct bias inputs. – Todd Feb 20 '16 at 9:44

It's a bit silly to start with a weak RNG (`System.Random`) and then carefully avoiding any bias in your own code. The expression "polishing a turd" comes to mind. – CodesInChaos Feb 20 '16 at 11:03

@CodesInChaos And now the apprentice has surpassed his master – Todd Feb 20 '16 at 12:02

2

Now in one-liner flavour.

```
private string RandomName
    {
        get
        {
            return new string(
                Enumerable.Repeat("ABCDEFGHIJKLMNOPQRSTUVWXYZ", 13)
                    .Select(s =>
                    {
                        var cryptoResult = new byte[4];
                        using (var cryptoProvider = new RNGCryptoSer
                            cryptoProvider.GetBytes(cryptoResult);
                        return s[new Random(BitConverter.ToInt32(cry
0)).Next(s.Length)];
                    })
                    .ToArray());
        }
    }
```

edited Nov 7 '18 at 6:57

piojo

3,642 12 24

answered Jan 22 '15 at 12:03

Matas Vaitkevicius
34.4k 16 169 176

1 Using a property for something that changes on every access is rather dubious. I'd recommend using a method instead. – CodesInChaos Jan 11 '16 at 16:34

1 `RNGCryptoServiceProvider` should be disposed after use. – Tsahi Asher Jan 10 '17 at 16:25

I fixed the IDisposable issue, but this is still highly dubious, creating a new RNGCryptoServiceProvider for each letter. – piojo Nov 7 '18 at 6:58

35

The main goals of my code are:

1. The distribution of strings is almost uniform (don't care about minor deviations, as long as they're small)
2. It outputs more than a few billion strings for each argument set. Generating an 8 character string (~47 bits of entropy) is meaningless if your PRNG only generates 2 billion (31 bits of entropy) different values.
3. It's secure, since I expect people to use this for passwords or other security tokens.

The first property is achieved by taking a 64 bit value modulo the alphabet size. For small alphabets (such as the 62 characters from the question) this leads to negligible bias. The second and third property are achieved by using `RNGCryptoServiceProvider` instead of `System.Random` .

```
using System;
using System.Security.Cryptography;

public static string GetRandomAlphanumericString(int length)
{
    const string alphanumericCharacters =
```

```
            "ABCDEFGHIJKLMNOPQRSTUVWXYZ" +
            "abcdefghijklmnopqrstuvwxyz" +
            "0123456789";
        return GetRandomString(length, alphanumericCharacters);
    }

    public static string GetRandomString(int length, IEnumerable<char> cl
    {
        if (length < 0)
            throw new ArgumentException("length must not be negative", "
        if (length > int.MaxValue / 8) // 250 million chars ought to be 
            throw new ArgumentException("length is too big", "length");
        if (characterSet == null)
            throw new ArgumentNullException("characterSet");
        var characterArray = characterSet.Distinct().ToArray();
        if (characterArray.Length == 0)
            throw new ArgumentException("characterSet must not be empty"

        var bytes = new byte[length * 8];
        var result = new char[length];
        using (var cryptoProvider = new RNGCryptoServiceProvider())
        {
            cryptoProvider.GetBytes(bytes);
        }
        for (int i = 0; i < length; i++)
        {
            ulong value = BitConverter.ToUInt64(bytes, i * 8);
            result[i] = characterArray[value % (uint)characterArray.Leng
        }
        return new string(result);
    }
```

edited Nov 7 '18 at 6:46

piojo
3,642 12 24

answered Nov 16 '12 at 11:57

CodesInChaos
90.1k 14 173 229

1 There is no intersection with 64 x Z and Math.Pow(2,Y). So while making bigger numbers reduces the bias, it doesn't eliminate it. I updated my

answer down below. my approach was to discard random inputs and

replace with another value. – Todd Feb 20 '16 at 10:00

@Todd I know that it doesn't eliminate the bias, but I chose the simplicity of this solution over eliminating a practically irrelevant bias. – CodesInChaos Feb 20 '16 at 10:32

I agree for most cases it's probably practically irrelevant. But now I've updated mine to be both as fast as Random and a little more secure than yours. All open source for all to share. Yes, I wasted way too much time on this... – Todd Feb 20 '16 at 12:06

If we are using RNG provider do we have any way to avoid bias in theory? I am not sure... If Todd mean the way when he generates additional random number (when we are in bias zone) then it can be wrong assumption. RNG has almost linear distribution of all generated values in average. But it does not mean that we will not have local correlation between generated bytes. So additional byte only for bias zone can still give us some bias but due to different reason. Most likely this bias will be very small. BUT in this case increasing of total generated bytes is more straightforward way. – Maxim Jul 31 '17 at 16:18

1 @Maxim You can use rejection to completely eliminate the bias (assuming the underlying generator is perfectly random). In exchange the code might run arbitrarily long (with exponentially small probability). – CodesInChaos Jul 31 '17 at 17:56

287

This implementation (found via google) looks sound to me.

Unlike some of the alternatives presented, this one is **cryptographically sound**.

```
using System.Security.Cryptography;
using System.Text;

namespace UniqueKey
{
    public class KeyGenerator
    {
        public static string GetUniqueKey(int size)
        {
```

```
            char[] chars =

"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".ToC
            byte[] data = new byte[size];
            using (RNGCryptoServiceProvider crypto = new RNGCryptoSe
            {
                crypto.GetBytes(data);
            }
            StringBuilder result = new StringBuilder(size);
            foreach (byte b in data)
            {
                result.Append(chars[b % (chars.Length)]);
            }
            return result.ToString();
        }
    }
}
```

Picked that one from a discussion of alternatives here

edited Nov 7 '18 at 6:43

piojo
3,642 12 24

answered Aug 27 '09 at 23:19

Eric J.
121k 47 270 497

10 This looks like the correct approach to me - random passwords, salts, entropy and so forth should not be generated using Random() which is optimized for speed and generates reproducible sequences of numbers; RNGCryptoServiceProvider.GetNonZeroBytes() on the other hand produces wild sequences of numbers that are NOT reproducible. – mindplay.dk Apr 18 '11 at 20:09

19 The letters are slightly biased(255 % 62 != 0). Despite this minor flaw, it is by far the best solution here. – CodesInChaos Mar 17 '12 at 20:52

11 Note that this is *not* sound if you want crypto-strength, unbiased randomness. (And if you don't want that then why use `RNGCSP` in the first place?) Using mod to index into the `chars` array means that you'll get

biased output unless `chars.Length` happens to be a divisor of 256. –

LukeH Apr 3 '12 at 16:35

13 One possibility to reduce the bias a lot, is requesting `4*maxSize` random bytes, then use `(UInt32)(BitConverter.ToInt32(data,4*i)% chars.Length` . I'd also use `GetBytes` instead of `GetNonZeroBytes` . And finally you can remove the first call to `GetNonZeroBytes` . You're not using its result. – CodesInChaos Apr 4 '12 at 8:19

7 Fun fact: A-Z a-z 0-9 is 62 characters. People are pointing out letter bias because 256 % 62 != 0. YouTube's video IDs are A-Z a-z 0-9, as well as "-" and "_", which produces 64 possible characters, which divides into 256 evenly. Coincidence? I think not! :) – qJake Sep 26 '16 at 18:25

17

One line of code `Membership.GeneratePassword()` does the trick :)

Here is a demo for the same.

edited May 3 '18 at 8:58

Stephen Kennedy

7,583 13 52 70

answered Jul 10 '14 at 20:40

Pooran

999 2 13 21

1 Looks like Microsoft moved the link.. another code sample is at msdn.microsoft.com/en-us/library/ms152017 or aspnet.4guysfromrolla.com/demos/GeneratePassword.aspx or developer.xamarin.com/api/member/… – Pooran Aug 2 '15 at 18:26

I thought of that but couldn't get rid of the non alphanumeric characters as the 2nd argument is the MINIMUM non alpha characters – ozzy432836 Apr 5 '18 at 10:29

My simple one line code works for me :)

6

```
string  random = string.Join("", Guid.NewGuid().ToString("n").Take(8

Response.Write(random.ToUpper());
Response.Write(random.ToLower());
```

To expand on this for any length string

```
public static string RandomString(int length)
{
    //length = length < 0 ? length * -1 : length;
    var str = "";

    do
    {
        str += Guid.NewGuid().ToString().Replace("-", "");
    }

    while (length > str.Length);

    return str.Substring(0, length);
}
```

edited Apr 5 '18 at 11:12

ozzy432836
1,494 1 19 23

answered Sep 29 '15 at 12:38

Raj kumar
494 6 11

I like the guid method too - feels really light – ozzy432836 Apr 5 '18 at 10:19

Solution 1 - largest 'range' with most flexible length

182

```
string get_unique_string(int string_length) {
    using(var rng = new RNGCryptoServiceProvider()) {
        var bit_count = (string_length * 6);
        var byte_count = ((bit_count + 7) / 8); // rounded up
        var bytes = new byte[byte_count];
        rng.GetBytes(bytes);
        return Convert.ToBase64String(bytes);
    }
}
```

This solution has more range than using a GUID because a GUID has a couple of fixed bits that are always the same and therefore not random, for example the 13 character in hex is always "4" - at least in a version 6 GUID.

This solution also lets you generate a string of any length.

Solution 2 - One line of code - good for up to 22 characters

```
Convert.ToBase64String(Guid.NewGuid().ToByteArray()).Substring(0, 8)
```

You can't generate strings as long as *Solution 1* and the string doesn't have the same range due to fixed bits in GUID's, but in a lot of cases this will do the job.

Solution 3 - Slightly less code

```
Guid.NewGuid().ToString("n").Substring(0, 8);
```

Mostly keeping this here for historical purpose. It uses slightly less code, that though comes as the expense of having less range - because it uses hex instead of base64 it takes more characters to represent the same range compared the other solutions.

Which means more chance of collision - testing it with 100,000 iterations of 8 character strings generated one duplicate.

edited Mar 20 '18 at 17:31

Luis Perez
23.5k 6 57 67

answered Aug 28 '09 at 0:00

Douglas
1,996 1 13 10

19 You actually generated a duplicate? Surprising at 5,316,911,983,139,663,491,615,228,241,121,400,000 possible combinations of GUIDs. – Alex Aug 28 '09 at 0:10

66 @Alex: He's shortening the GUID to 8 characters, so the probability of collisions is much higher than that of GUIDs. – dtb Aug 28 '09 at 0:19

19 Nobody can appreciate this other than nerds :) Yes you are absolutely right, the 8 char limit makes a difference. – Alex Aug 28 '09 at 0:36

29 Guid.NewGuid().ToString("n") will keep the dashes out, no Replace() call needed. But it should be mentioned, GUIDs are only 0-9 and A-F. The number of combinations is "good enough," but nowhere close to what a *true* alphanumeric random string permits. The chances of collision are 1:4,294,967,296 -- the same as a random 32-bit integer. – richardtallent Aug 28 '09 at 0:40

30 1) GUIDs are designed to be unique, not random. While current versions of windows generate V4 GUIDs which are indeed random, that's not guaranteed. For example older versions of windows used V1 GUIDs, where your could would fail. 2) Just using hex characters reduces the quality of the random string significantly. From 47 to 32 bits. 3) People are underestimating the collision probability, since they give it for individual pairs. If you generate 100k 32 bit values, you probably have one collision among them. See Birthday problem. – CodesInChaos Nov 16 '12 at 12:33

Just some performance comparisons of the various answers in this thread:

19

Methods & Setup

```
// what's available
public static string possibleChars = "abcdefghijklmnopqrstuvwxyz";
// optimized (?) what's available
public static char[] possibleCharsArray = possibleChars.ToCharArray(
// optimized (precalculated) count
public static int possibleCharsAvailable = possibleChars.Length;
// shared randomization thingy
public static Random random = new Random();


// http://stackoverflow.com/a/1344242/1037948
public string LinqIsTheNewBlack(int num) {
    return new string(
    Enumerable.Repeat(possibleCharsArray, num)
              .Select(s => s[random.Next(s.Length)])
              .ToArray());
}

// http://stackoverflow.com/a/1344258/1037948
public string ForLoop(int num) {
    var result = new char[num];
    while(num-- > 0) {
        result[num] = possibleCharsArray[random.Next(possibleCharsAv
    }
    return new string(result);
}

public string ForLoopNonOptimized(int num) {
    var result = new char[num];
    while(num-- > 0) {
        result[num] = possibleChars[random.Next(possibleChars.Length
    }
    return new string(result);
}

public string Repeat(int num) {
    return new string(new char[num].Select(o =>
possibleCharsArray[random.Next(possibleCharsAvailable)]).ToArray());
}

// http://stackoverflow.com/a/1518495/1037948
public string GenerateRandomString(int num) {
  var rBytes = new byte[num];
  random.NextBytes(rBytes);
  var rName = new char[num];
  while(num-- > 0)
```

```
    rName[num] = possibleCharsArray[rBytes[num] % possibleCharsAvail
  return new string(rName);
}

//SecureFastRandom - or SolidSwiftRandom
static string GenerateRandomString(int Length) //Configurable output
{
    byte[] rBytes = new byte[Length];
    char[] rName = new char[Length];
    SolidSwiftRandom.GetNextBytesWithMax(rBytes, biasZone);
    for (var i = 0; i < Length; i++)
    {
        rName[i] = charSet[rBytes[i] % charSet.Length];
    }
    return new string(rName);
}
```

Results

Tested in LinqPad. For string size of 10, generates:

> - from Linq = chdgmevhcy [10]
> - from Loop = gtnoaryhxr [10]
> - from Select = rsndbztyby [10]
> - from GenerateRandomString = owyefjjakj [10]
> - from SecureFastRandom = VzougLYHYP [10]
> - from SecureFastRandom-NoCache = oVQXNGmO1S [10]

And the performance numbers tend to vary slightly, very occasionally `NonOptimized` is actually faster, and sometimes `ForLoop` and `GenerateRandomString` switch who's in the lead.

> - LinqIsTheNewBlack (10000x) = 96762 ticks elapsed (9.6762 ms)
> - ForLoop (10000x) = 28970 ticks elapsed (2.897 ms)
> - ForLoopNonOptimized (10000x) = 33336 ticks elapsed (3.3336 ms)

- Repeat (10000x) = 78547 ticks elapsed (7.8547 ms)
- GenerateRandomString (10000x) = 27416 ticks elapsed (2.7416 ms)
- SecureFastRandom (10000x) = 13176 ticks elapsed (5ms) lowest [Different machine]
- SecureFastRandom-NoCache (10000x) = 39541 ticks elapsed (17ms) lowest [Different machine]

edited Feb 21 '16 at 11:50

Todd
10.4k 4 27 43

answered Jun 13 '13 at 16:45

drzaus
15.7k 7 90 149

3 Would be interesting to know which ones created dupes. – Rebecca Aug 1 '13 at 12:43

@Junto -- to figure out which results in duplicates, something like `var many = 10000; Assert.AreEqual(many, new bool[many].Select(o => EachRandomizingMethod(10)).Distinct().Count());` , where you replace `EachRandomizingMethod` with...each method – drzaus Sep 11 '13 at 17:06

thank you @Todd for updating the comparisons with your newer answer – drzaus Feb 22 '16 at 19:15

You just use the assembly `SRVTextToImage` . And write below code to generate random string.

-3

```
CaptchaRandomImage c1 = new CaptchaRandomImage();
            string text = c1.GetRandomString(8);
```

Mostly it is used to implement the Captcha. But in your case it also works. Hope it helps.

answered Feb 16 '16 at 19:05

Salman Mushtaq
227 1 16

So you're saying we should use some 3rd party library originally designed to generate CAPTCHAs to generate a random string? – Tsahi Asher Jan 10 '17 at 16:30

3 The string isn't the only thing that's random about this suggestion! i love it, definately the most original! – BanksySan Jan 12 '17 at 18:27

1

I don't know how cryptographically sound this is, but it's more readable and concise than the more intricate solutions by far (imo), and it should be more "random" than `System.Random` -based solutions.

```
return alphabet
    .OrderBy(c => Guid.NewGuid())
    .Take(strLength)
    .Aggregate(
        new StringBuilder(),
        (builder, c) => builder.Append(c))
    .ToString();
```

I can't decide if I think this version or the next one is "prettier", but they give the exact same results:

```
return new string(alphabet
    .OrderBy(o => Guid.NewGuid())
    .Take(strLength)
    .ToArray());
```

Granted, it isn't optimized for speed, so if it's mission critical to generate millions of random strings every second, try another one!

NOTE: This solution doesn't allow for repetitions of symbols in the alphabet, and the alphabet MUST be of equal or greater size than the output string, making this approach less desirable in some circumstances, it all depends on your use-case.

edited Jan 14 '16 at 18:14

answered Jan 14 '16 at 18:08

sara
2,330 7 23

4

```
 public static string RandomString(int length)
    {
        const string chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
        var random = new Random();
        return new string(Enumerable.Repeat(chars, length).Select(s
s[random.Next(s.Length)]).ToArray());
    }
```

answered Nov 24 '15 at 8:46

Tejas
347 3 12

Here is a variant of Eric J's solution, i.e. cryptographically sound, for WinRT (Windows Store App):

2

```
public static string GenerateRandomString(int length)
{
    var chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY
    var result = new StringBuilder(length);
    for (int i = 0; i < length; ++i)
    {
        result.Append(CryptographicBuffer.GenerateRandomNumber() % cl
    }
    return result.ToString();
}
```

If performance matters (especially when length is high):

```
public static string GenerateRandomString(int length)
{
    var chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY
    var result = new System.Text.StringBuilder(length);
    var bytes = CryptographicBuffer.GenerateRandom((uint)length * 4)
    for (int i = 0; i < bytes.Length; i += 4)
    {
        result.Append(BitConverter.ToUInt32(bytes, i) % chars.Length
    }
    return result.ToString();
}
```

edited Oct 27 '15 at 14:20

answered Oct 26 '15 at 14:46

huyc

114 1 7

1 This is *not* cryptographically sound. There is a small bias due to the modulus operation not spreading the entire width of ulong equally into 62 chars. – Lie Ryan May 30 '17 at 15:24

0

Here is a mechanism to generate a random alpha-numeric string (I use this to generate passwords and test data) without defining the alphabet and numbers,

CleanupBase64 will remove necessary parts in the string and keep adding random alpha-numeric letters recursively.

```
public static string GenerateRandomString(int length)
{
    var numArray = new byte[length];
    new RNGCryptoServiceProvider().GetBytes(numArray);
    return CleanUpBase64String(Convert.ToBase64String(numArr
}

private static string CleanUpBase64String(string input, int
{
    input = input.Replace("-", "");
    input = input.Replace("=", "");
    input = input.Replace("/", "");
    input = input.Replace("+", "");
    input = input.Replace(" ", "");
    while (input.Length < maxLength)
        input = input + GenerateRandomString(maxLength);
    return input.Length <= maxLength ?
        input.ToUpper() : //In my case I want capital letter
        input.ToUpper().Substring(0, maxLength);
}
```

edited Sep 30 '15 at 2:55

answered Sep 30 '15 at 2:43

Dhanuka777

4,771 3 45 95

You have declared `GenerateRandomString` and make a call to `GetRandomString` from within `SanitiseBase64String`. Also you have declared `SanitiseBase64String` and call `CleanUpBase64String` in `GenerateRandomString`. – Wai Ha Lee Sep 30 '15 at 2:53

1 Thanks, I have corrected that. – Dhanuka777 Sep 30 '15 at 3:02

10

The code written by Eric J. is quite sloppy (it is quite clear that it is from 6 years ago... he probably wouldn't write that code today), and there are even some problems.

> Unlike some of the alternatives presented, this one is cryptographically sound.

Untrue... There is a bias in the password (as written in a comment), `bcdefgh` are a little more probable than the others (the `a` isn't because by the `GetNonZeroBytes` it isn't generating bytes with a value of zero, so the bias for the `a` is balanced by it), so it isn't really cryptographically sound.

This should correct all the problems.

```
public static string GetUniqueKey(int size = 6, string chars =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890")
{
    using (var crypto = new RNGCryptoServiceProvider())
    {
        var data = new byte[size];

        // If chars.Length isn't a power of 2 then there is a bias i
        // we simply use the modulus operator. The first characters
        // chars will be more probable than the last ones.

        // buffer used if we encounter an unusable random byte. We w
        // regenerate it in this buffer
        byte[] smallBuffer = null;

        // Maximum random number that can be used without introducin
        // bias
        int maxRandom = byte.MaxValue - ((byte.MaxValue + 1) % chars

        crypto.GetBytes(data);

        var result = new char[size];
```

```
            for (int i = 0; i < size; i++)
            {
                byte v = data[i];

                while (v > maxRandom)
                {
                    if (smallBuffer == null)
                    {
                        smallBuffer = new byte[1];
                    }

                    crypto.GetBytes(smallBuffer);
                    v = smallBuffer[0];
                }

                result[i] = chars[v % chars.Length];
            }

            return new string(result);
        }
    }
```

answered Aug 12 '15 at 7:53

xanatos

90.1k 7 142 205

2

A solution without using `Random` :

```
var chars =
Enumerable.Repeat("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvw

var randomStr = new string(chars.SelectMany(str => str)
                                .OrderBy(c => Guid.NewGuid())
                                .Take(8).ToArray());
```

edited Aug 11 '15 at 10:02

answered Aug 11 '15 at 9:42

w.b
9,065 4 21 39

2 NewGuid uses random internally. So this is still using random, it's just hiding it. – Wedge May 27 '16 at 16:50

-1

Very simple solution. It uses ASCII values and just generates "random" characters in between them.

```
public static class UsernameTools
{
    public static string GenerateRandomUsername(int length = 10)
    {
        Random random = new Random();
        StringBuilder sbuilder = new StringBuilder();
        for (int x = 0; x < length; ++x)
        {
            sbuilder.Append((char)random.Next(33, 126));
        }
        return sbuilder.ToString();
    }

}
```

answered Jun 21 '15 at 22:46

Serguei Fedorov
4,413 9 44 81

Try to combine two parts: unique (sequence, counter or date) and random

3

```
public class RandomStringGenerator
{
    public static string Gen()
    {
        return ConvertToBase(DateTime.UtcNow.ToFileTimeUtc()) + GenR
//keep length fixed at least of one part
    }

    private static string GenRandomStrings(int strLen)
    {
        var result = string.Empty;

        var Gen = new RNGCryptoServiceProvider();
        var data = new byte[1];

        while (result.Length < strLen)
        {
            Gen.GetNonZeroBytes(data);
            int code = data[0];
            if (code > 48 && code < 57 || // 0-9
                code > 65 && code < 90 || // A-Z
                code > 97 && code < 122   // a-z
                )
            {
                result += Convert.ToChar(code);
            }
        }

        return result;
    }

    private static string ConvertToBase(long num, int nbase = 36)
    {
        var chars = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"; //if you
more secure - change order of letter here

        // check if we can convert to another base
        if (nbase < 2 || nbase > chars.Length)
            return null;

        int r;
        var newNumber = string.Empty;

        // in r we have the offset of the char that was converted to
        while (num >= nbase)
        {
```

```
            r = (int) (num % nbase);
            newNumber = chars[r] + newNumber;
            num = num / nbase;
        }
        // the last number to convert
        newNumber = chars[(int)num] + newNumber;

        return newNumber;
    }
}
```

Tests:

```
[Test]
    public void Generator_Should_BeUnigue1()
    {
        //Given
        var loop = Enumerable.Range(0, 1000);
        //When
        var str = loop.Select(x=> RandomStringGenerator.Gen());
        //Then
        var distinct = str.Distinct();
        Assert.AreEqual(loop.Count(),distinct.Count()); // Or
Assert.IsTrue(distinct.Count() < 0.95 * loop.Count())
    }
```

edited Jun 15 '15 at 7:51

CodesInChaos

90.1k 14 173 229

answered Jan 1 '15 at 19:08

RouR

4,299 1 23 21

code and a test... gotta like it – JasonCoder Jul 24 '15 at 16:20

1) You can use character literals instead of the ASCII values associated with those characters. 2) You have an off-by-one mistake in your interval matching code. You need to use `<=` and `>=` instead of `<` and `>`. 3) I'd add the unnecessary parentheses around the `&&` expressions to make it clear they have precedence, but of course that's only a stylistic choice. – CodesInChaos Oct 24 '16 at 16:58

+ 1 Good for removing bias and adding testing. I'm not sure why you prepend your random string with a timestamp-derived string though? Also, you still need to dispose your RNGCryptoServiceProvider – monty Jun 7 '18 at 22:11

5

A slightly cleaner version of DTB's solution.

```
var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
var random = new Random();
var list = Enumerable.Repeat(0, 8).Select(x=>chars[random.Next(cl
return string.Join("", list);
```

Your style preferences may vary.

answered Apr 27 '15 at 20:47

Rob Deary
687 8 3

This is much better and more efficient than the accepted answer. – Wedge May 27 '16 at 16:49

6

Question: Why should I waste my time using `Enumerable.Range` instead of typing in `"ABCDEFGHJKLMNOPQRSTUVWXYZ0123456789"` ?

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Test
{
    public static void Main()
```

```
    {
        var randomCharacters = GetRandomCharacters(8, true);
        Console.WriteLine(new string(randomCharacters.ToArray()));
    }

    private static List<char> getAvailableRandomCharacters(bool incl
    {
        var integers = Enumerable.Empty<int>();
        integers = integers.Concat(Enumerable.Range('A', 26));
        integers = integers.Concat(Enumerable.Range('0', 10));

        if ( includeLowerCase )
            integers = integers.Concat(Enumerable.Range('a', 26));

        return integers.Select(i => (char)i).ToList();
    }

    public static IEnumerable<char> GetRandomCharacters(int count, b
includeLowerCase)
    {
        var characters = getAvailableRandomCharacters(includeLowerCa
        var random = new Random();
        var result = Enumerable.Range(0, count)
            .Select(_ => characters[random.Next(characters.Count)]);

        return result;
    }
}
```

Answer: Magic strings are BAD. Did ANYONE notice there was no " `I` " in my string at the top? My mother taught me not to use magic strings for this very reason...

n.b. 1: As many others like @dtb said, don't use `System.Random` if you need cryptographic security...

n.b. 2: This answer isn't the most efficient or shortest, but I wanted the space to separate the answer from the question. The purpose of my answer is more to warn against magic strings than to provide a fancy innovative answer.

edited Feb 22 '15 at 18:17

answered Feb 22 '15 at 15:16

Wai Ha Lee

6,135 12 41 66

Why do I care that there's no " `I` ?" – Hill Dec 12 '16 at 19:01

1 Alphanumeric (ignoring case) is `[A-Z0-9]` . If, by accident, your random string only ever covers `[A-HJ-Z0-9]` the result doesn't cover the full allowable range, which may be problematic. – Wai Ha Lee Dec 13 '16 at 16:14

How would that be problematic? So it doesn't contain `I` . Is it because there's one less character and that makes it easier to crack? What are the statistics on crackable passwords that contain 35 characters in the range than 36. I think I would rather risk it... or just proof reed the character range... than include all that extra garbage in my code. But, that's me. I mean, not to be a butt hole, I'm just saying. Sometimes I think programmers have a tendency go the extra-complex route for the sake of being extra-complex. – Hill Dec 13 '16 at 16:22

1 It deepends on the use case. It's very common to exclude characters such as `I` and `O` from these types of random strings to avoid humans confusing them with `1` and `0` . If you don't care about having a human-readable string, fine, but if it's something someone might need to type, then it's actually smart to remove those characters. – Chris Pratt Aug 22 '17 at 3:04

4

I was looking for a more specific answer, where I want to control the format of the random string and came across this post. For example: license plates (of cars) have a specific format (per country) and I wanted to created random license plates.
I decided to write my own extension method of Random for this. (this is in order to reuse the same Random object, as you could have doubles in multi-threading scenarios). I created a gist

(https://gist.github.com/SamVanhoutte/808845ca78b9c041e928), but will also copy the extension class here:

```
void Main()
{
    Random rnd = new Random();
    rnd.GetString("1-###-000").Dump();
}

public static class RandomExtensions
{
    public static string GetString(this Random random, string format
    {
        // Based on http://stackoverflow.com/questions/1344221/how-c
random-alphanumeric-strings-in-c
        // Added logic to specify the format of the random string (#
string, 0 will be random numeric, other characters remain)
        StringBuilder result = new StringBuilder();
        for(int formatIndex = 0; formatIndex < format.Length ; forma
        {
            switch(format.ToUpper()[formatIndex])
            {
                case '0': result.Append(getRandomNumeric(random)); b
                case '#': result.Append(getRandomCharacter(random));
                default : result.Append(format[formatIndex]); break;
            }
        }
        return result.ToString();
    }

    private static char getRandomCharacter(Random random)
    {
        string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        return chars[random.Next(chars.Length)];
    }

    private static char getRandomNumeric(Random random)
    {
        string nums = "0123456789";
        return nums[random.Next(nums.Length)];
    }
}
```

answered Jan 22 '15 at 10:39

Sam Vanhoutte


1,674 9 20

1

I know this one is not the best way. But you can try this.

```
string str = Path.GetRandomFileName(); //This method returns a rando
characters
str = str.Replace(".","");
Console.WriteLine("Random string: " + str);
```

answered Jul 15 '14 at 12:42


Sagar
1,439 4 16 29

The best ONE-LINE approach! – DenisKolodin Jan 8 '17 at 9:28

2 How is that one line? Console.WriteLine($"Random string: {Path.GetRandomFileName().Replace(".","")}"); is one line. – PmanAce May 23 '18 at 15:02

6

We also use custom string random but we implemented is as a string's helper so it provides some flexibility...

```
public static string Random(this string chars, int length = 8)
{
    var randomString = new StringBuilder();
    var random = new Random();

    for (int i = 0; i < length; i++)
        randomString.Append(chars[random.Next(chars.Length)]);

    return randomString.ToString();
}
```

Usage

```
var random = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".Random();
```

or

```
var random =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789".Ran
```

answered Jun 26 '14 at 15:12

Mr. Pumpkin

3,276 3 30 47

28

The simplest:

```
public static string GetRandomAlphaNumeric()
{
    return Path.GetRandomFileName().Replace(".", "").Substring(0, 8)
}
```

You can get better performance if you hard code the char array and rely on `System.Random` :

```
public static string GetRandomAlphaNumeric()
{
    var chars = "abcdefghijklmnopqrstuvwxyz0123456789";
    return new string(chars.Select(c =>
chars[random.Next(chars.Length)]).Take(8).ToArray());
}
```

If ever you worry the English alphabets can change sometime around and you might lose business, then you can avoid hard coding, but should perform slightly worse (comparable to `Path.GetRandomFileName` approach)

```
public static string GetRandomAlphaNumeric()
{
    var chars = 'a'.To('z').Concat('0'.To('9')).ToList();
    return new string(chars.Select(c =>
chars[random.Next(chars.Length)]).Take(8).ToArray());
}

public static IEnumerable<char> To(this char start, char end)
{
    if (end < start)
        throw new ArgumentOutOfRangeException("the end char should n
start char", innerException: null);
    return Enumerable.Range(start, end - start + 1).Select(i => (cha
}
```

The last two approaches looks better if you can make them an extension method on `System.Random` instance.

edited Jun 4 '14 at 10:35

answered Apr 13 '13 at 7:09

nawfal

43.9k 36 258 305

1 Using `chars.Select` is a big ugly since it relies on the output size being at most the alphabet size. – CodesInChaos Jun 4 '14 at 10:09

@CodesInChaos I'm not sure if I understand you. You mean in the `'a'.To('z')` approach? – nawfal Jun 4 '14 at 10:18

1 1) `chars.Select()` .Take(n)` only works if `chars.Count >= n` . Selecting on a sequence you don't actually use is a bit unintuitive, especially with that implicit length constraint. I'd rather use `Enumerable.Range` or `Enumerable.Repeat` . 2) The error message "the end char should be less than start char" is the wrong way round/missing a `not` . – CodesInChaos Jun 4 '14 at 10:20

@CodesInChaos but in my case `chars.Count` is way `> n` . Also I dont get the unintuitive part. That does make all uses `Take` unintuitive doesnt it? I dont believe it. Thanks for pointing the typo. – nawfal Jun 4 '14 at

10:34

4 This is featured on theDailyWTF.com as a CodeSOD article. – user177800 Oct 3 '18 at 5:33

0

```
public static class StringHelper
{
    private static readonly Random random = new Random();

    private const int randomSymbolsDefaultCount = 8;
    private const string availableChars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

    private static int randomSymbolsIndex = 0;

    public static string GetRandomSymbols()
    {
        return GetRandomSymbols(randomSymbolsDefaultCount);
    }

    public static string GetRandomSymbols(int count)
    {
        var index = randomSymbolsIndex;
        var result = new string(
            Enumerable.Repeat(availableChars, count)
                      .Select(s => {
                          index += random.Next(s.Length);
                          if (index >= s.Length)
                              index -= s.Length;
                          return s[index];
                      })
                      .ToArray());
        randomSymbolsIndex = index;
        return result;
    }
}
```

edited Apr 7 '14 at 8:08

answered Apr 7 '14 at 6:25

KregHEk
319 3 11

2 1) static methods should be thread safe. 2) What's the point of incrementing the index instead of using the result of `random.Next` directly? Complicates the code and doesn't achieve anything useful. – CodesInChaos Jun 4 '14 at 10:01

6

Another option could be to use Linq and aggregate random chars into a stringbuilder.

```
var chars = "abcdefghijklmnopqrstuvwxyz123456789".ToArray();
string pw = Enumerable.Range(0, passwordLength)
                      .Aggregate(
                          new StringBuilder(),
                          (sb, n) => sb.Append((chars[random.Next(ch
                          sb => sb.ToString());
```

edited Dec 8 '12 at 20:39

answered Jan 24 '12 at 23:42

AAD
261 3 9

0

If your values are not completely random, but in fact may depend on something - you may compute an md5 or sha1 hash of that 'somwthing' and then truncate it to whatever length you want.

Also you may generate and truncate a guid.

answered Mar 29 '10 at 15:48

Alexey B.
708 3 14

Horrible, I know, but I just couldn't help myself:

4

```
namespace ConsoleApplication2
{
    using System;
    using System.Text.RegularExpressions;

    class Program
    {
        static void Main(string[] args)
        {
            Random adomRng = new Random();
            string rndString = string.Empty;
            char c;

            for (int i = 0; i < 8; i++)
            {
                while
(!Regex.IsMatch((c=Convert.ToChar(adomRng.Next(48,128))).ToString(),
                rndString += c;
            }

            Console.WriteLine(rndString + Environment.NewLine);
        }
    }
}
```

edited Aug 28 '09 at 0:37

answered Aug 28 '09 at 0:26

james
834 5 7

57

Here's an example that I stole from Sam Allen example at Dot Net Perls

If you only need 8 characters, then use Path.GetRandomFileName() in the System.IO namespace. Sam says using the "Path.GetRandomFileName method here is sometimes superior, because it uses RNGCryptoServiceProvider for better randomness. However, it is limited to 11 random characters."

GetRandomFileName always returns a 12 character string with a period at the 9th character. So you'll need to strip the period (since that's not random) and then take 8 characters from the string. Actually, you could just take the first 8 characters and not worry about the period.

```
public string Get8CharacterRandomString()
{
    string path = Path.GetRandomFileName();
    path = path.Replace(".", ""); // Remove period.
    return path.Substring(0, 8);  // Return 8 character string
}
```

PS: thanks Sam

edited Aug 27 '09 at 23:37

dtb
173k 27 347 404

answered Aug 27 '09 at 23:36

Adam Porad
11.2k 2 27 51

3 A bit too clever for my tastes. I prefer the more straightforward approach. – JohnFx Aug 28 '09 at 0:39

2 @RaifAtef: Can you please explain what you meant by "wont work in medium trust"? Thank you. – Animesh Apr 10 '12 at 18:45

2 I stand corrected, it seems that it doesn't require full trust. I misread it for Path.GetTempFileName() which calls into "Environment" methods which don't work on medium trust. Medium trust: stackoverflow.com/questions/2617454/… – Raif Atef Apr 11 '12 at 13:45

22 This works well. I ran it through 100,000 iterations and never had a duplicate name. However, I **did** find several vulgar words (in English). Wouldn't have even thought of this except one of the first ones in the list had F*** in it . Just a heads up if you use this for something the user will see. – techturtle Oct 9 '12 at 22:35

3 @techturtle Thanks for the warning. I suppose there's a risk for vulgar words with any random string generation that uses all letters in the alphabet. – Adam Porad Oct 10 '12 at 23:38