

Forum:[Hack Forums](#)

Thread: [Generate Random 24 Byte String](#)

draxyee - December 28, 2019, 5:26 am

Example: ??????_?Y;?

My code:

```
Code    [STAThread]
static void Main(string[] args)
{
    while (true)
    {
        string twenty4 = GetRandom24ByteString();
        Console.WriteLine(twenty4);
        Clipboard.SetText(twenty4);

        Console.Read();
    }
}

public static string GetRandom24ByteString()
{
    var rnd = new ThreadLocal<Random>(() => new
Random(Guid.NewGuid().GetHashCode()));
    byte[] b = new byte[24]; rnd.Value.NextBytes(b);
    return Encoding.UTF8.GetString(b, 0, b.Length);
}
```

However, when I paste it into byte count, it is:

or

I don't know what I'm doign wrong ?

Anyone help? Thanks!

MavenGradle - December 28, 2019, 10:54 am

You're doing quite a few things wrong. First of all, I don't know why you need a Code while (true) loop in your code. The only purpose it serves is to make it impossible to quit the process gracefully.

Second, the byte counter you're using is actually not a byte counter but a character byte counter. If you're converting your bytes (which don't hold any encoding due to the way you're generating them)

into a specific encoding such as ASCII or UTF8, you are effectively unable to display certain byte values, because for example the ASCII character sets don't handle all characters and byte values.

There's also the 'small' matter of how many bytes it requires to represent a character. UTF8, for example, requires minimum one byte, but if the value is bigger than 127 then UTF8 may take multiple bytes (2, 3 or 4 bytes since it's variable width). That is evident, for example, by the display of those question mark 'place holder' characters. What you are getting, is an incomplete or 'broken' representation, if you can even call it that. I'd like to call it the result of programmer that doesn't know what he's doing.

If you take the following byte value for example:

Code 0xFF You will not get a correct UTF8 string because it's value is much larger than the maximum of 127 (0x7F) for a single byte.

Third, if you count the length of the byte array, you'll notice that it really is 24 bytes because you set it to be a maximum of 24 bytes long:

Code `byte[] b = new byte[24];`

If you were to attempt to set byte at index 25, you'd get an `IndexOutOfRangeException`:

Code `data[25] = 0x0;`

Go ahead and run that and see for yourself. So, now that we're absolutely positive that your byte array is in fact 24 bytes long, what else are you doing wrong? Well, you're not using OOP standards. This might be overkill in some scenario's, but just for the sake of following good coding principles, let me fix that for you:

Code using System;

namespace ByteTest

```
{
    public sealed class ByteGenerator
    {
        private readonly Random _random;

        /// <summary>
        /// Initialize a new ByteGenerator
        /// </summary>
        /// <param name="seed">The seed that can be used to initialize a Random object</param>
        public ByteGenerator(int seed)
        {
            _random = new Random(seed);
        }

        /// <summary>
        /// Initialize a new ByteGenerator
        /// </summary>
        public ByteGenerator()
        {
            _random = new Random();
        }

        /// <summary>
        /// Generate an array of random bytes with a given length
        /// </summary>
        /// <param name="length">The length of the byte array that should be generated</param>
```

```

/// <returns>A byte array that contains random bytes with a given length</returns>
public byte[] GenerateBytes(int length)
{
    if (length <= 0) throw new ArgumentOutOfRangeException(nameof(length));

    byte[] data = new byte[length];
    _random.NextBytes(data);
    return data;
}
}

```

What this class does, is allow you to reuse the same code, in different environments. As you can see the class is sealed which means it cannot be inherited by other classes.

Next up. How do you properly display those bytes? Well, there are a few ways in which you can get a correct representation of said bytes. The first way to do it would be to adhere to the character sets, but then again you'd possibly be generating more (or less) bytes than your desired amount of bytes which beats the purpose of your question. While [ASCII](#) and [UTF8](#) are representations or character sets, you are not adhering to their (or any) standards, hence the weird place holder characters and missing or extra characters. What I recommend that you do, instead, is to display them in hexadecimal format, by using the [BitConverter](#) class:

Code `ByteGenerator generator = new ByteGenerator();` // You can use initialize using a seed as well, if required.

`BitConverter.ToString(generator.GenerateBytes(24));`

This will output something similar to:

Code `26-5F-65-B2-C6-0F-1D-7C-D0-DB-10-95-7A-DA-BF-DA-89-18-9C-13-B0-D3-13-6A`

If you know how the [hexadecimal positional system](#) works, you can clearly see that there are exactly 24 bytes in there, which is exactly what you're asking for. Your code essentially generates 24 bytes as well and the way they're being counted is 'illogical' because you're not thinking straight in regards to the representation of said bytes and conversion mechanics. For example, the hexadecimal representation of the bytes above is invalid UTF8.

This is all a very simplified explanation, but you should understand that your code is actually generating 24 bytes, like you're asking it to do. I would recommend that you seriously consider looking more into character sets and what they actually are. Here's some reading material for you:

https://en.wikipedia.org/wiki/Character_encoding

<http://kunststube.net/encoding/>

<https://javarevisited.blogspot.com/2015/02/difference-between-utf-8-utf-16-and-utf.html>

Fun fact: Something you may or may not be aware of, is that the default string in the .NET Framework (<=4.8 at the time of writing) is encoded in UTF16.

Eternal Darkness - December 28, 2019, 2:44 pm

Depending on use may want a better random:

[System.Security.Cryptography.RandomNumberGenerator](#) (note: Slower).

Seeding is good, but there are better randomization techniques. Likely out of scope of discussion but OP might as well make the call of desired amount of randomization.

This may offer better assistance in determining make-up of string btw (hover, over the purple string for example): <https://www.soscisurvey.de/tools/view-chars.php>

(Dec 28, 2019 11:54 AM)MavenGradle Wrote: [\[CITE\]](#) Code ByteGenerator generator = new ByteGenerator(); // Seed = non-constant, rather-"unique" value
BitConverter.ToString(generator.GenerateBytes(24));
Other than a ToString() method doesn't get much cleaner than that (I understand the return value and everything, just saying).
Adjusted comment, inb4 I see: ByteGenerator(24)

(Dec 28, 2019 11:54 AM)MavenGradle Wrote: [\[CITE\]](#) I would recommend that you seriously consider looking more into character sets and what they actually are.
Yep. OP, research this. Read through the rest of MavenGradle's response, but research this...

Plenty of information programmers should know about a char, notably the [ASCII table](#) and the extended characters gained via different Encodings. Maven was meaning the same just elaborating on the basic subject matter. I do rather like the UTF-8 wiki page as well:
<https://en.wikipedia.org/wiki/UTF-8>.

Would be curious to know what the use for these characters are, you would be better off using character sets from another language/something rather than using non-printable characters/whatnot in most cases. You should make sure you have a range for these characters/some limitation. Not sure what the use is, but I will say that certain characters are meant for certain things, and should not be used whenever you feel like.

Another thing:
<https://stackoverflow.com/questions/5750203/how-to-write-unicode-characters-to-the-console>