How do I encode and decode a base64 string?

Ask Question

713

1. How do I return a base64 encoded string given a string?
2. How do I decode a base64 encoded string into a string?

c# base64

156

asked Jul 31 '12 at 15:06
Kevin Driedger
26.4k 15 41 53

4 If this is a "sharing the knowledge" question and answer, I think we're looking for something a bit more in-depth. Also a quick search of SO turns up: stackoverflow.com/a/7368168/419 – Kev Aug 1 '12 at 1:46

1 @Gnark Any string is encoded by a certain underlying bit-encoding schema. Be it ASCII, UTF7, UTF8, ... The question posed is at best incomplete. – Lorenz Lo Sauer Dec 4 '13 at 21:17

2 Ask yourself do you really need to do this? Remember base64 is primarily intended for representing binary data in ASCII, for storing in a char field in a database or sending via email (where new lines could be injected). Do you really want to take character data, convert it to bytes, then convert it back to character data, this time unreadable and with no hint of what the original encoding was ? – bbsimonbb Apr 14 '16 at 8:22

Why should we care about the original encoding? We encode the string into the bytes using UTF8 representation, which can represent all the possible string characters. We then serialize that data and on the other end we deserialize that data and we reconstruct the same string that we originally had (string object doesn't hold the information about encoding used anyway). So why is there any concern related to the encoding used? We can consider it like a proprietary way of representing the serialized data, which we shouldn't be interested at anyway. – Mladen B. Mar 12 at 11:08

5 Answers

-3

To decode

```
from base64 import b64decode
data = input('Enter a base64 string with padding: ')
output = b64decode(data).hex()
print(output)
```

This is for python 3 (only tested on 3.7 and 3.5, but it should work for the others)

edited Feb 27 at 0:31

answered Feb 24 at 1:31

Eric Jin
6 4

Why the ".hex()"? – U. Windl Feb 24 at 2:01

You need the .hex() if you want it converted into a hexadecimal string. `>>> b64decode('Hello+World=')` (output) `b'\x1d\xe9e\xa3\xe5\xa8\xaeW'` but after adding .hex(): `>>> b64decode('Hello+World=').hex()` (output) `'1de965a3e5a8ae57'` – Eric Jin Feb 24 at 18:30

Actually the request was to convert some string to BASE64 and back. So if the original was some binary string, the result should be the same binary string IMHO (not a hex string). That's why I was asking. – U. Windl Feb 25 at 20:26

24

Based on the answers by Andrew Fox and Cebe, I turned it around and made them string extensions instead of Base64String extensions.

```
public static class StringExtensions
{
    public static string ToBase64(this string text)
    {
        return ToBase64(text, Encoding.UTF8);
    }

    public static string ToBase64(this string text, Encoding encodin

    {
        if (string.IsNullOrEmpty(text))
        {
            return text;
        }

        byte[] textAsBytes = encoding.GetBytes(text);
        return Convert.ToBase64String(textAsBytes);
    }

    public static bool TryParseBase64(this string text, out string d
    {
        return TryParseBase64(text, Encoding.UTF8, out decodedText);
    }

    public static bool TryParseBase64(this string text, Encoding enc
decodedText)
    {
        if (string.IsNullOrEmpty(text))
        {
            decodedText = text;
            return false;
        }

        try
        {
            byte[] textAsBytes = Convert.FromBase64String(text);
            decodedText = encoding.GetString(textAsBytes);
            return true;
        }
        catch (Exception)
        {
            decodedText = null;
            return false;
        }
    }
}
```

edited Jun 19 '18 at 9:56

Azeem
3,109 4 12 24

answered Mar 6 '18 at 13:25

j2associates
534 3 12

1364

Encode

```
public static string Base64Encode(string plainText) {
  var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(plainText
  return System.Convert.ToBase64String(plainTextBytes);
}
```

Decode

```
public static string Base64Decode(string base64EncodedData) {
  var base64EncodedBytes = System.Convert.FromBase64String(base64Er
  return System.Text.Encoding.UTF8.GetString(base64EncodedBytes);
}
```

edited Jun 19 '18 at 9:55

Azeem
3,109 4 12 24

answered Jul 31 '12 at 15:06

Kevin Driedger
26.4k 15 41 53

37 Null checks for input strings in both functions and the solution is perfect :) – Sverrir Sigmundarson Mar 29 '14 at 23:05

20 @SverrirSigmundarson: That or make them extension methods. – T.J. Crowder Dec 30 '14 at 12:38

64 @SverrirSigmundarson - Why do a null check? He's not the one dereferencing the input string. Null checks should prevent `NullReferenceException` in your own code, not somebody else's. – ken Feb 2 '15 at 18:44

13 @ken And somebody else will say "you should only expose errors in your own code, not somebody else's", invoking the principle of least surprise, spiced with "fail early" and "proper encapsulation". Sometimes this means wrapping errors of lower-level components, sometimes something else entirely. In this case, I'll agree that wrapping a deref error is definitely dubious (plus we're all slowly agreeing to the fact that null as a concept is a bit of a hack to begin with), but we can still see some effects otherwise: the parameter name given in the exception might not be correct if left unchecked. – tne Aug 19 '15 at 8:57

6 return System.Text.Encoding.UTF8.GetString(base64EncodedBytes, 0, base64EncodedBytes.Length); for windows phone 8 – steveen zoleko Dec 9 '15 at 17:26

20

A slight variation on andrew.fox answer, as the string to decode might not be a correct base64 encoded string:

```
using System;

namespace Service.Support
{
    public static class Base64
    {
        public static string ToBase64(this System.Text.Encoding enco
        {
            if (text == null)
            {
                return null;
            }

            byte[] textAsBytes = encoding.GetBytes(text);
            return Convert.ToBase64String(textAsBytes);
        }

        public static bool TryParseBase64(this System.Text.Encoding
encodedText, out string decodedText)
        {
            if (encodedText == null)
            {
                decodedText = null;
                return false;
            }

            try
            {
                byte[] textAsBytes = Convert.FromBase64String(encode
                decodedText = encoding.GetString(textAsBytes);
                return true;
            }
            catch (Exception)
            {
                decodedText = null;
                return false;
            }
        }
    }
}
```

answered Apr 13 '16 at 8:05

Cebe
439 5 10

I'm sharing my implementation with some neat features:

40

- uses Extension Methods for Encoding class. Rationale is that someone may need to support different types of encodings (not only UTF8).
- Another improvement is failing gracefully with null result for null entry - it's very useful in real life scenarios and supports equivalence for X=decode(encode(X)).

Remark: Remember that to use Extension Method you **have to** (!) import the namespace with `using` keyword (in this case `using MyApplication.Helpers.Encoding`).

Code:

```
namespace MyApplication.Helpers.Encoding
{
    public static class EncodingForBase64
    {
        public static string EncodeBase64(this System.Text.Encoding 
text)
        {
            if (text == null)
            {
                return null;
            }

            byte[] textAsBytes = encoding.GetBytes(text);
            return System.Convert.ToBase64String(textAsBytes);
        }

        public static string DecodeBase64(this System.Text.Encoding 
encodedText)
        {
            if (encodedText == null)
            {
                return null;
            }

            byte[] textAsBytes = System.Convert.FromBase64String(enc
            return encoding.GetString(textAsBytes);
        }
    }
}
```

Usage example:

```
using MyApplication.Helpers.Encoding; // !!!

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Test1();
            Test2();
        }

        static void Test1()
        {
            string textEncoded = System.Text.Encoding.UTF8.EncodeBas
            System.Diagnostics.Debug.Assert(textEncoded == "dGVzdDEu

            string textDecoded = System.Text.Encoding.UTF8.DecodeBas
            System.Diagnostics.Debug.Assert(textDecoded == "test1...
        }

        static void Test2()
        {
            string textEncoded = System.Text.Encoding.UTF8.EncodeBas
            System.Diagnostics.Debug.Assert(textEncoded == null);

            string textDecoded = System.Text.Encoding.UTF8.DecodeBas
            System.Diagnostics.Debug.Assert(textDecoded == null);
        }
    }
}
```

answered Mar 17 '16 at 20:46

andrew.fox
3,737 4 34 56

Returning `null` in case of `null` is a very inconsistent behaviour. No other .net API that works with strings does that. – t3chb0t Jun 22 '18 at 6:10

2 @t3chb0t feel free to adjust it to your needs. As the way it's presented here was adjusted to ours. This is not a public API ;) – andrew.fox Jun 22

'18 at 9:35

Don't you now have to send 2 variables to the other party in your communication (to whom you're sending base64 encoded data)? You need to send both the encoding used and the actual base64 data? Isn't it easier if you use a convention on both sides to use the same encoding? That way you'd only have to send base64 data, right? – Mladen B. Mar 12 at 11:12