Dotneteers.net

All for .net, .net for all!

# Parsing C# string literals

The C# language specification treats in details how we can specify a literal string constant in the source code. This literal may contain escape sequences and also can be a verbatim string.

It could be useful if we could use the syntax in external files like an application configuration file to read and process the string constants on the way the C# compiler does. For example, we could have the following string constants in a file:

```
"\\Hello, world"
"\\\a\b\'\"\0\f \t\v"
"Hello\x1\x12\x123\x1234"
```

After processing the information above we get the following strings (I put the control character names between braces, but actually their Unicode value is in the memory):

```
\Hello, world
\{alert}{backspace}'"{null}{formfeed}{newline}{carriage return}{tab}{vertical tab}
Hello{0x0001}{0x0012}{0x0123}{0x1234}
```

I created a utility class to implement this functionality (I used it in my LINQ over C# project), it is used by the syntax parser. In this post I publish the source code of this class, I hope some of you will find it useful.

```
using System;
using System.Globalization;
using System.Text;

namespace CSharpParser.Utility
{
  // ==================================================================================
  /// <summary>
  /// This static class involves helper methods that use strings.
  /// </summary>
  // ==================================================================================
  public static class StringHelper
  {
    // --------------------------------------------------------------------------------
    /// <summary>
    /// Converts a C# literal string into a normal string.
    /// </summary>
    /// <param name="source">Source C# literal string.</param>
    /// <returns>
    /// Normal string representation.
    /// </returns>
    // --------------------------------------------------------------------------------
    public static string StringFromCSharpLiteral(string source)
    {
      StringBuilder sb = new StringBuilder(source.Length);
      int pos = 0;
      while (pos < source.Length)
      {
        char c = source[pos];
        if (c == '\\')
        {
          // --- Handle escape sequences
          pos++;
          if (pos >= source.Length) throw new ArgumentException("Missing escape sequence");
          switch (source[pos])
          {
            // --- Simple character escapes
            case '\'': c = '\''; break;
            case '\"': c = '\"'; break;
            case '\\': c = '\\'; break;
            case '0': c = '\0'; break;
            case 'a': c = '\a'; break;
            case 'b': c = '\b'; break;
            case 'f': c = '\f'; break;
            case 'n': c = ' '; break;
            case 'r': c = ' '; break;
            case 't': c = '\t'; break;
            case 'v': c = '\v'; break;
            case 'x':
              // --- Hexa escape (1-4 digits)
              StringBuilder hexa = new StringBuilder(10);
              pos++;
              if (pos >= source.Length)
                throw new ArgumentException("Missing escape sequence");
              c = source[pos];
              if (Char.IsDigit(c) || (c >= 'a' && c <= 'f') || (c >= 'A' && c <= 'F'))
              {
                hexa.Append(c);
                pos++;
                if (pos < source.Length)
                {
                  c = source[pos];
                  if (Char.IsDigit(c) || (c >= 'a' && c <= 'f') || (c >= 'A' && c <= 'F'))
                  {
                    hexa.Append(c);
                    pos++;
                    if (pos < source.Length)
                    {
                      c = source[pos];
                      if (Char.IsDigit(c) || (c >= 'a' && c <= 'f') ||
                        (c >= 'A' && c <= 'F'))
                      {
                        hexa.Append(c);
                        pos++;
                        if (pos < source.Length)
                        {
                          c = source[pos];
                          if (Char.IsDigit(c) || (c >= 'a' && c <= 'f') ||
                            (c >= 'A' && c <= 'F'))
                          {
                            hexa.Append(c);
                            pos++;
                          }
                        }
                      }
                    }
                  }
                }
              }
              c = (char)Int32.Parse(hexa.ToString(), NumberStyles.HexNumber);
              pos--;
              break;
            case 'u':
              // Unicode hexa escape (exactly 4 digits)
              pos++;
              if (pos + 3 >= source.Length)
                throw new ArgumentException("Unrecognized escape sequence");
              try
              {
                uint charValue = UInt32.Parse(source.Substring(pos, 4),
                  NumberStyles.HexNumber);
                c = (char)charValue;
                pos += 3;
              }
              catch (SystemException)
            {
              throw new ArgumentException("Unrecognized escape sequence");
            }
            break;
          case 'U':
            // Unicode hexa escape (exactly 8 digits, first four must be 0000)
            pos++;
            if (pos + 7 >= source.Length)
              throw new ArgumentException("Unrecognized escape sequence");
            try
            {
              uint charValue = UInt32.Parse(source.Substring(pos, 8),
                NumberStyles.HexNumber);
              if (charValue > 0xffff)
                throw new ArgumentException("Unrecognized escape sequence");
              c = (char)charValue;
              pos += 7;
            }
            catch (SystemException)
            {
              throw new ArgumentException("Unrecognized escape sequence");
            }
            break;
          default:
            throw new ArgumentException("Unrecognized escape sequence");
        }
      }
      pos++;
      sb.Append(c);
    }
    return sb.ToString();
  }

  // ------------------------------------------------------------------------------------
  /// <summary>
  /// Converts a C# verbatim literal string into a normal string.
  /// </summary>
  /// <param name="source">Source C# literal string.</param>
  /// <returns>
  /// Normal string representation.
  /// </returns>
  // ------------------------------------------------------------------------------------
  public static string StringFromVerbatimLiteral(string source)
  {
    StringBuilder sb = new StringBuilder(source.Length);
    int pos = 0;
    while (pos < source.Length)
    {
      char c = source[pos];
      if (c == '\"')
      {
        // --- Handle escape sequences
        pos++;
        if (pos >= source.Length) throw new ArgumentException("Missing escape sequence");
        if (source[pos] == '\"') c = '\"';
        else throw new ArgumentException("Unrecognized escape sequence");
      }
      pos++;
      sb.Append(c);
    }
    return sb.ToString();
  }

  // ------------------------------------------------------------------------------------
  /// <summary>
  /// Converts a C# literal string into a normal character..
  /// </summary>
  /// <param name="source">Source C# literal string.</param>
  /// <returns>
  /// Normal char representation.
```

```
    /// </returns>
    // ----------------------------------------------------------------------------------
    public static char CharFromCSharpLiteral(string source)
    {
      string result = StringFromCSharpLiteral(source);
      if (result.Length != 1)
        throw new ArgumentException("Invalid char literal");
      return result[0];
    }
  }
}
```

If you would like to use it in your own code, you can check the its behavior with the following unit tests:

```
using System;
using CSharpParser.Utility;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace CSharpParserTest
{
  /// <summary>
  /// Summary description for StringHelperTest
  /// </summary>
  [TestClass]
  public class StringHelperTest
  {
    [TestMethod]
    public void StringLiteralsAreOK()
    {
      string csharp = "Hello, world";
      string normal = "Hello, world";
      Assert.AreEqual(StringHelper.StringFromCSharpLiteral(csharp), normal);
      csharp = "\\\\Hello, world";
      normal = "\\Hello, world";
      Assert.AreEqual(StringHelper.StringFromCSharpLiteral(csharp), normal);
      csharp = "\\\\\\a\\b\\'\\\"\\0\\f\ \ \\t\\v";
      normal = "\\\a\b\'\"\0\f \t\v";
      Assert.AreEqual(StringHelper.StringFromCSharpLiteral(csharp), normal);
      csharp = "Hello\\x1\\x12\\x123\\x1234";
      normal = "Hello\x1\x12\x123\x1234";
      Assert.AreEqual(StringHelper.StringFromCSharpLiteral(csharp), normal);
      csharp = "Hello\\x1\\x12\\x123\\x123456";
      normal = "Hello\x1\x12\x123\x123456";
      Assert.AreEqual(StringHelper.StringFromCSharpLiteral(csharp), normal);
      csharp = "\\u1234\\u2345\\u3456";
      normal = "\u1234\u2345\u3456";
      Assert.AreEqual(StringHelper.StringFromCSharpLiteral(csharp), normal);
      csharp = "\\U00001234\\U00002345\\U00003456";
      normal = "\U00001234\U00002345\U00003456";
      Assert.AreEqual(StringHelper.StringFromCSharpLiteral(csharp), normal);
    }

    [TestMethod]
    public void StringLiteralExceptionsAreOK()
    {
      int count = 0;
      string csharp = "\\";
      try {StringHelper.StringFromCSharpLiteral(csharp); }
      catch (ArgumentException) { count++; }

      csharp = "\\q";
      try { StringHelper.StringFromCSharpLiteral(csharp); }
      catch (ArgumentException) { count++; }

      csharp = "\\x";
      try { StringHelper.StringFromCSharpLiteral(csharp); }
      catch (ArgumentException) { count++; }

      csharp = "\\u123";
      try { StringHelper.StringFromCSharpLiteral(csharp); }
      catch (ArgumentException) { count++; }
      Assert.AreEqual(count, 4);

      csharp = "\\U123345";
      try { StringHelper.StringFromCSharpLiteral(csharp); }
      catch (ArgumentException) { count++; }

      csharp = "\\U00012345";
      try { StringHelper.StringFromCSharpLiteral(csharp); }
      catch (ArgumentException) { count++; }

      csharp = "\\U00002345";
      try { StringHelper.StringFromCSharpLiteral(csharp); }
      catch (ArgumentException) { count++; }
      Assert.AreEqual(count, 6);
    }

    [TestMethod]
    public void CharLiteralsAreOK()
    {
      string csharp = "\\x020";
      char normal = '\x020';
      Assert.AreEqual(StringHelper.CharFromCSharpLiteral(csharp), normal);
      csharp = "\ ";
      normal = ' ';
      Assert.AreEqual(StringHelper.CharFromCSharpLiteral(csharp), normal);
    }

    [TestMethod]
    public void CharLiteralExceptionsAreOK()
    {
      int count = 0;
      string csharp = "\\";
      try { StringHelper.CharFromCSharpLiteral(csharp); }
      catch (ArgumentException) { count++; }

      csharp = "\\q";
      try { StringHelper.CharFromCSharpLiteral(csharp); }
      catch (ArgumentException) { count++; }

      csharp = "\\x";
      try { StringHelper.CharFromCSharpLiteral(csharp); }
      catch (ArgumentException) { count++; }

      csharp = "\\x020ABC";
      try { StringHelper.CharFromCSharpLiteral(csharp); }
      catch (ArgumentException) { count++; }

      Assert.AreEqual(count, 4);
    }

    [TestMethod]
    public void VerbatimLiteralsAreOK()
    {
      string csharp = @"Hello, world";
      string normal = "Hello, world";
      Assert.AreEqual(StringHelper.StringFromVerbatimLiteral(csharp), normal);
      csharp = "\"\"Hello, world\"\"";
      normal = @"""Hello, world""";
      Assert.AreEqual(StringHelper.StringFromVerbatimLiteral(csharp), normal);
    }
  }
}
```

Should you find a bug, I would be very happy if you'd contact me. If you wrote a test case to indicate the bug would help me lot. I hope, you find this utility useful ☺.

Posted Aug 03 2008, 08:21 AM by inovak
Filed under: C#, .NET

## Comments

**Martin S** wrote **re: Parsing C# string literals**
on Mon, Aug 17 2009 16:26

Hi Istvan,

I am wondering whether the following is correct:

```
    case 'n': c = ' '; break;

    case 'r': c = ' '; break;
```

It is mapping \n and \r to a space; but I thought that these three chars (\n \r and space) denote different bytes, so should be decoded differently.

Thanks! -M

**chris** wrote **re: Parsing C# string literals**
on Wed, Oct 7 2009 9:12

As far as I know for .NET the

Regex..::.Unescape Method

does (at least most of) the job

**Alex** wrote **re: Parsing C# string literals**
on Tue, Nov 3 2009 11:42

Seems very good , thanks !

**Alex** wrote **re: Parsing C# string literals**
on Tue, Nov 3 2009 11:43

by the way Regex.Unescape does not do exactly the same thing.. It also converts other characters and some others it doesn't.

**<a href="http://www.blackitsoft.com/inventory-pos-software.aspx" title=" Inventory POS System" target="blank"> Inventory POS System </a>** wrote **re: Parsing C# string literals**
on Wed, Dec 22 2010 6:26

I like your article and it really gives an outstanding idea that is very helpful for all the people on web.

**buyviagra online** wrote **re: Parsing C# string literals**
on Sat, Feb 23 2013 19:06

gErfEW Im obliged for the article.Really thank you! Much obliged.