Exercise - Character Escape Sequences and Verbatim Strings

10 minutes

Suppose you've been asked by your manager to create a mockup of a command-line tool that will generate invoices in both English and Japanese. You don't have to build the actual functionality that generates the invoices yet. You only need to provide the command line's user interface to internal customers in the billing department for their approval. Your manager asked you to make sure you add formatting to make the current progress of the tool clear. Your manager also asked you to provide instructions for the Japanese users on how to generate invoices in Japanese.

Formatting Literal Strings in C#

C# provides a wealth of options for formatting strings. We'll only look at a few of the most used in this module. However just about anything you can imagine, you could accomplish. The hardest part may be knowing what terminology to use when searching for answers.

Character Escape Sequences

An **escape character sequence** is a special instruction to the runtime that you want to insert a special character that will affect the output of your string. In C#, the escape character sequences begin with a backslash `\` and then include another character. For example, the `\n` sequence will add a new line, and a `\t` sequence will add a tab.

The following code uses escape character sequences to add whitespace.

C# Copy Run

```
Console.WriteLine("Hello\nWorld!");
Console.WriteLine("Hello\tWorld!");
```

If you run the code, you'll see the following output.

Output Copy

```
Hello
World!
Hello   World!
```

What if you need to insert a double-quotation mark in a literal string? If you don't use the character escape sequence, you'll confuse the compiler because it will think you want to terminate the string prematurely ... and will not understand the purpose of the characters after the second double-quotation mark.

C# Copy Run

```
Console.WriteLine("Hello "World"!");
```

The .NET Editor will put a red squiggly line under `World`. But if you attempt to run the code anyway, you would see the following output.

Output Copy

```
(1,27): error CS1003: Syntax error, ',' expected
(1,27): error CS0103: The name 'World' does not exist in the current context
(1,32): error CS1003: Syntax error, ',' expected
```

To handle that situation, use the `\"` escape sequence.

C# Copy Run

```
Console.WriteLine("Hello \"World\"!");
```

If you run the code above, you would see the following output.

Output Copy

```
Hello "World"!
```

What if you need to use the backslash for other purposes, like to display a file path?

C# Copy Run

```
Console.WriteLine("c:\source\repos");
```

Unfortunately, C# reserves the backslash for escape sequences, so if you run the code, the compiler will display the following error.

Output Copy

```
(1,22): error CS1009: Unrecognized escape sequence
```

The problem is the sequence `\s`. The `\r` doesn't produce an error because it is a valid escape sequence for a carriage return. However, it's unlikely that you would want to use a carriage return in this context.

To solve the problem, you use the `\\` to display a single backslash.

C# Copy Run

```csharp
Console.WriteLine("c:\\source\\repos");
```

Escaping the back slash character produces the output you intended.

Output Copy

```
c:\source\repos
```

Step 1 - Format the output of the command-line application using character escape sequences

To create the mockup of our command line tool, add the following code in the editor.

C# Copy Run

```csharp
Console.WriteLine("Generating invoices for customer \"ABC Corp\" ...\n");
Console.WriteLine("Invoice: 1021\t\tComplete!");
Console.WriteLine("Invoice: 1022\t\tComplete!");
Console.WriteLine("\nOutput Directory:\t");
```

You should see the following output.

Output Copy

```
Generating invoices for customer "ABC Corp" ...

Invoice: 1021       Complete!
Invoice: 1022       Complete!

Output Directory:
```

Verbatim String Literal

A verbatim string literal will keep all whitespace and characters without the need to escape the backslash. To create a verbatim string, use the `@` directive before the literal string. Two consecutive double-quote characters (`""`) are printed as a single double-quote character (`"`) in the output string.

C# Copy Run

```
Console.WriteLine(@"    c:\source\repos
      (""this is where your code goes"")");
```

Notice that as written in the `@` directive, the string spans two lines, the whitespace is kept, and two consecutive double-quote characters are applied as a single double-quote character in the following output:

Output Copy

```
   c:\source\repos
      ("this is where your code goes")
```

Step 2 - Format the output of the command-line application using a verbatim literal string

Add the following line of code beneath the code that was added in step 1.

C# Copy Run

```
Console.Write(@"c:\invoices");
```

When you run the code, you should see now the following output that includes the "output directory" instruction.

Output Copy

```
Generating invoices for customer "ABC Corp" ...

Invoice: 1021       Complete!
Invoice: 1022       Complete!

Output Directory:
c:\invoices
```

Unicode Escape Characters

You can also add encoded characters in literal strings using the `\u` escape sequence, then a four-character code representing some character in Unicode (UTF-16).

C# Copy Run

```csharp
// Kon'nichiwa World
Console.WriteLine("\u3053\u3093\u306B\u3061\u306F World!");
```

Note

There are several caveats here. First, some consoles like the Windows Command Prompt will not display all Unicode characters. It will replace those characters with question mark characters instead. Also, the examples used here are UTF-16. Some characters require UTF-32 and therefore require a different escape sequence. This is a complicated subject, and this module is only aiming at showing you what is possible. Depending on your need, you may need to spend quite a bit of time learning and working with Unicode characters in your applications.

Step 3 - Format the output of the command-line application using unicode escape characters

To complete the mocked up command-line user interface, we'll add a phrase in Japanese that translates to "To generate Japanese invoices", then provides a verbatim literal string with the application executable with a flag. We'll also add some escape sequences for formatting.

Add the following code to your application.

C# Copy Run

```csharp
// To generate Japanese invoices:
// Nihon no seikyū-sho o seisei suru ni wa:
Console.Write("\n\n\u65e5\u672c\u306e\u8acb\u6c42\u66f8\u3092\u751f\u6210\u3059\u3
08b\u306b\u306f\uff1a\n\t");
Console.WriteLine(@"c:\invoices\app.exe -j");
```

The entire code example should look like the following.

C# Copy Run

```csharp
Console.WriteLine("Generating invoices for customer \"ABC Corp\" ...\n");
Console.WriteLine("Invoice: 1021\t\tComplete!");
Console.WriteLine("Invoice: 1022\t\tComplete!");
Console.WriteLine("\nOutput Directory:\t");
Console.Write(@"c:\invoices");

// To generate Japanese invoices:
// Nihon no seikyū-sho o seisei suru ni wa:
Console.Write("\n\n\u65e5\u672c\u306e\u8acb\u6c42\u66f8\u3092\u751f\u6210\u3059\u3
08b\u306b\u306f\uff1a\n\t");
Console.WriteLine(@"c:\invoices\app.exe -j");
```

When you run the code, you should see the following output.

Output Copy

```
Generating invoices for customer "ABC Corp" ...

Invoice: 1021       Complete!
Invoice: 1022       Complete!

Output Directory:
c:\invoices

日本の請求書を生成するには：
    c:\invoices\app.exe -j
```

Recap

Here's the most important items to remember about formatting literal strings:

- Use character escape sequences when you need to insert a special character into a literal string, like a tab `\t`, new line `\n`, or a double quotation mark `\"`.
- Use an escape character for the backslash `\\` when you need to use a backslash in all other scenarios.
- Use the `@` directive to create a verbatim string literal that keeps all whitespace formatting and backslash characters in a string.
- Use the `\u` plus a four-character code to represent Unicode characters (UTF-16) in a string.
- Unicode characters may not print out correctly depending on the application.

Next unit: Exercise - String Concatenation

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

English (United States) Theme

.NET Editor

Press CTRL + M, TAB to exit the editor

Clear Run

Output