C# String escape sequence

Escape sequence	Character name
\'	Single quote
\"	Double quote
\\	Backslash
\0	Null
\a	Alert
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Veritcal tab

Reflection it

C# Numeric & Char Literals

object x = 10;	int
object x = 0x0A;	int (hexadecimal)
object x = 10L;	long
object x = 10U;	Uint
object x = 10UL;	Ulong
object x = 10.5M;	Decimal
object x = 10.5F;	Float
object x = 10.5D;	double
object x = 'X'	char
object x = '/x0058'	char (hexadecimal)

String Formatting

The format parameter is embedded with zero or more format specifications of the form, `{N [, M][: formatString ]}` where:

- N is a zero-based integer indicating the argument to be formatted.
- M is an optional integer indicating the width of the region to contain the formatted value, padded with spaces. If the sign of M is negative, the formatted value is left-justified in the region; if the sign of M is positive, the value is right-justified.
- formatString is an optional string of formatting codes

Sample:

```
CultureInfo ci = CultureInfo.CreateSpecificCulture("en-US");
string s = string.Format(ci, "|{0,12:C}|\r\n{1:D}", 4321.04, new DateTime(2003, 9, 14));
```

Result:

```
|   $4,321.04|
Sunday, September 14, 2003
```

Standard Numeric Format Strings

Standard numeric format strings are used to return common numeric string types. A standard format string takes the form Axx where A is an alphabetic character called the format specifier, and xx is a sequence of digits called the precision specifier.

Format specifier	Description
C or c	Currency
D or d	Decimal (decimal integer)
E or e	Exponent
F or f	Fixed point
G or g	General
N or n	Number
P or p	Percentage
R or r	Round-trip (for floating-point values only); guarantees that a numeric value converted to a string will be parsed back into the same numeric value
X or x	Hex

Sample:
```
int i = 123456;
Console.WriteLine("{0:C}", i);    // € 123,456.00
Console.WriteLine("{0:C4}", i);   // € 123,456.0000
Console.WriteLine("{0:D}", i);    // 123456
```

Custom Numeric Format Strings

0	Display zero placeholder
#	Display digit placeholder
.	Decimal point
,	Group separator
%	Percent notation
E+0, E-0, e+0, e-0	Exponent notation
;	Section separator, specifies different output if the numeric value to be formatted is positive, negative, or zero

Sample:
```
Console.WriteLine("{0:#0.00;#0.00-;<zero>}", 123456);       // 123456.00
Console.WriteLine("{0:#0.00;#0.00-;<zero>}", -123456);      // 123456.00-
Console.WriteLine("{0:#0.00;#0.00-;<zero>}", 0);            // <zero>
```

Standard DateTime Format Strings

d	Short date pattern
D	Long date pattern
t	Short time pattern
T	Long time pattern
f	Full date/time pattern (short time)
F	Full date/time pattern (long time)
g	General date/time pattern (short time)
G	General date/time pattern (long time)
M	Month day pattern
s	Sortable date/time pattern; conforms to ISO 8601
U	Universal sortable date/time pattern
Y	Year month pattern

Custom DateTime Format Strings

d, dd, ddd, dddd	Days: 'single' digit, double digit, abbreviated name, full name
M, MM, MMM, MMMM	Month: 'single' digit, double digit, abbreviated name, full name
y, yy, yyyy	Years: 'single' digit, double digit, four digit
g or gg	Displays the era (A.D. for example)
h, hh, H, HH	Hours: 1-12 'single' digit, 1-12 double digit,1-23 'single' digit, 1-23 double digit
m, mm	Minutes: 0-59 'single' digit, 0-59 double digit
s, ss	Seconds: 0-59 'single' digit, 0-59 double digit
f, ff, fff, ... , ffffff	Displays seconds fractions: one digit, two digits, three digits, … , zeven digits
t	First character of the A.M./P.M. designator
:	Time separator
/	Date separator
g or gg	Displays the era (A.D. for example)

Regular Expressions

Character	Description
\	Marks the next character as a special character, a literal, a backreference, or an octal escape. For example, 'n' matches the character "n". '\n' matches a newline character. The sequence '\\' matches "\" and "\(" matches "(".
^	Matches the position at the beginning of the input string. If the RegExp object's Multiline property is set, ^ also matches the position following '\n' or '\r'.
$	Matches the position at the end of the input string. If the RegExp object's Multiline property is set, $ also matches the position preceding '\n' or '\r'.
*	Matches the preceding character or subexpression zero or more times. For example, zo* matches "z" and "zoo". * is equivalent to {0,}.
+	Matches the preceding character or subexpression one or more times. For example, 'zo+' matches "zo" and "zoo", but not "z". + is equivalent to {1,}.
?	Matches the preceding character or subexpression zero or one time. For example, "do(es)?" matches the "do" in "do" or "does". ? is equivalent to {0,1}
{n}	n is a nonnegative integer. Matches exactly n times. For example, 'o{2}' does not match the 'o' in "Bob," but matches the two o's in "food".
{n,}	n is a nonnegative integer. Matches at least n times. For example, 'o{2,}' does not match the "o" in "Bob" and matches all the o's in "foooood". 'o{1,}' is equivalent to 'o+'. 'o{0,}' is equivalent to 'o*'.
{n,m}	m and n are nonnegative integers, where n <= m. Matches at least n and at most m times. For example, "o{1,3}" matches the first three o's in "fooooood". 'o{0,1}' is equivalent to 'o?'. Note that you cannot put a space between the comma and the numbers.
?	When this character immediately follows any of the other quantifiers (*, +, ?, {n}, {n,}, {n,m}), the matching pattern is non-greedy. A non-greedy pattern matches as little of the searched string as possible, whereas the default greedy pattern matches as much of the searched string as possible. For example, in the string "oooo", 'o+?' matches a single "o", while 'o+' matches all 'o's.
.	Matches any single character except "\n". To match any character including the '\n', use a pattern such as '[\s\S].
(pattern)	Matches pattern and captures the match. The captured match can be retrieved from the resulting Matches collection, using the SubMatches collection in VBScript or the $0…$9 properties in JScript. To match parentheses characters ( ), use '\(' or '\)'.
(?:pattern)	Matches pattern but does not capture the match, that is, it is a non-capturing match that is not stored for possible later use. This is useful for combining parts of a pattern with the "or" character (|). For example, 'industr(?:y|ies) is a more economical expression than 'industry|industries'.
(?=pattern)	Positive lookahead matches the search string at any point where a string matching pattern begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?=95|98|NT|2000)' matches "Windows" in "Windows 2000" but not "Windows" in "Windows 3.1". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
(?!pattern)	Negative lookahead matches the search string at any point where a string not matching pattern begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?!95|98|NT|2000)' matches "Windows" in "Windows 3.1" but does not match "Windows" in "Windows 2000". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
x|y	Matches either x or y. For example, 'z|food' matches "z" or "food". '(z|f)ood' matches "zood" or "food".
[xyz]	A character set. Matches any one of the enclosed characters. For example, '[abc]' matches the 'a' in "plain".
[^xyz]	A negative character set. Matches any character not enclosed. For example, '[^abc]' matches the 'p' in "plain".
[a-z]	A range of characters. Matches any character in the specified range. For example, '[a-z]' matches any lowercase alphabetic character in the range 'a' through 'z'.
[^a-z]	A negative range characters. Matches any character not in the specified range. For example, '[^a-z]' matches any character not in the range 'a' through 'z'.
\b	Matches a word boundary, that is, the position between a word and a space. For example, 'er\b' matches the 'er' in "never" but not the 'er' in "verb".
\B	Matches a nonword boundary. 'er\B' matches the 'er' in "verb" but not the 'er' in "never".

\cx	Matches the control character indicated by x. For example, \cM matches a Control-M or carriage return character. The value of x must be in the range of A-Z or a-z. If not, c is assumed to be a literal 'c' character.
\d	Matches a digit character. Equivalent to [0-9].
\D	Matches a nondigit character. Equivalent to [^0-9].
\f	Matches a form-feed character. Equivalent to \x0c and \cL.
\n	Matches a newline character. Equivalent to \x0a and \cJ.
\r	Matches a carriage return character. Equivalent to \x0d and \cM.
\s	Matches any white space character including space, tab, form-feed, and so on. Equivalent to [\f\n\r\t\v].
\S	Matches any non-white space character. Equivalent to [^ \f\n\r\t\v].
\t	Matches a tab character. Equivalent to \x09 and \cI.
\v	Matches a vertical tab character. Equivalent to \x0b and \cK.
\w	Matches any word character including underscore. Equivalent to '[A-Za-z0-9_]'.
\W	Matches any nonword character. Equivalent to '[^A-Za-z0-9_]'.
\xn	Matches n, where n is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, '\x41' matches "A". '\x041' is equivalent to '\x04' & "1". Allows ASCII codes to be used in regular expressions.
\num	Matches num, where num is a positive integer. A reference back to captured matches. For example, '(.)\1' matches two consecutive identical characters.
\n	Identifies either an octal escape value or a backreference. If \n is preceded by at least n captured subexpressions, n is a backreference. Otherwise, n is an octal escape value if n is an octal digit (0-7).
\nm	Identifies either an octal escape value or a backreference. If \nm is preceded by at least nm captured subexpressions, nm is a backreference. If \nm is preceded by at least n captures, n is a backreference followed by literal m. If neither of the preceding conditions exist, \nm matches octal escape value nm when n and m are octal digits (0-7).
\nml	Matches octal escape value nml when n is an octal digit (0-3) and m and l are octal digits (0-7).
\un	Matches n, where n is a Unicode character expressed as four hexadecimal digits. For example, \u00A9 matches the copyright symbol (©).