

## VisualAPL Text Literals

- Text literals in VisualAPL include the string and the char[] (array of characters) data types
  - There are two types of C# string literals:
    - regular with format: “...”
    - verbatim with format: @”...”
  - The C# char[] (character array literal) is of the format ‘...’, however the verbatim @ prefix does not apply to the char[] data type in C# and the content of the ‘...’ assignment by reference can specify only one character at a time in C#.
- Escape sequences may be used in regular string literals and character array literals, for example:
  - \x#### for hexadecimal
  - \?, where ? are special character identifiers, e.g. \"(quotation mark), \'(single quote), \r (return), \n(newline), etc.
  - \u#### for Unicode
- To include “ in the content of a verbatim string literal, use the format: @”...””, i.e. include double “” in the content.
- Escape sequences within verbatim string literals are not considered as escape sequences, e.g. @“\r” represents the text (\r) and not the return character.
- Besides supporting all the .Net data types including string and char[] and the associated C# syntax, VisualAPL also supports the cvar (Cielo Variable) data type to implement the dynamic data typing of VisualAPL
- In VisualAPL because there are two types of assignment, by value (←) and by reference (=), their use by the programmer will determine the data type of the result of an assignment:
  - When the programmer assigns a variable by value (←), the result of the GetType() method on that variable will be cvar. The “...” or ‘...’ format may be used with assignment by value.
  - When the programmer assigns a text literal variable by reference (=), the result of the GetType() method on that variable will be:
    - string if the assignment uses the “...” or @”...” format
    - char[] if the assignment uses the ‘...’ format.
  - VisualAPL extends the char[] data type assignment by reference with any number of characters specified in the content of ‘...’.
  - VisualAPL extends the char[] data type assignment by reference using the verbatim @ prefix with the format var\_name = @’...’. The result of the GetType() method of the var\_name variable will be char[].

Study the following examples closely and try them out in the Cielo Explorer:

```
Cielo Explorer for Visual Studio - Professional v1
Cielo 1.0.7001
Copyright © VisualCielo. All rights reserved.
clear session

    T="abcd" A Assignment by reference
    T
'abcd
    T.GetType()
System.String

    T1<-"abcd" A Assignment by value
    T1
'abcd
    T1.GetType()
VisualCielo.Numeric.DataObjects.cvar

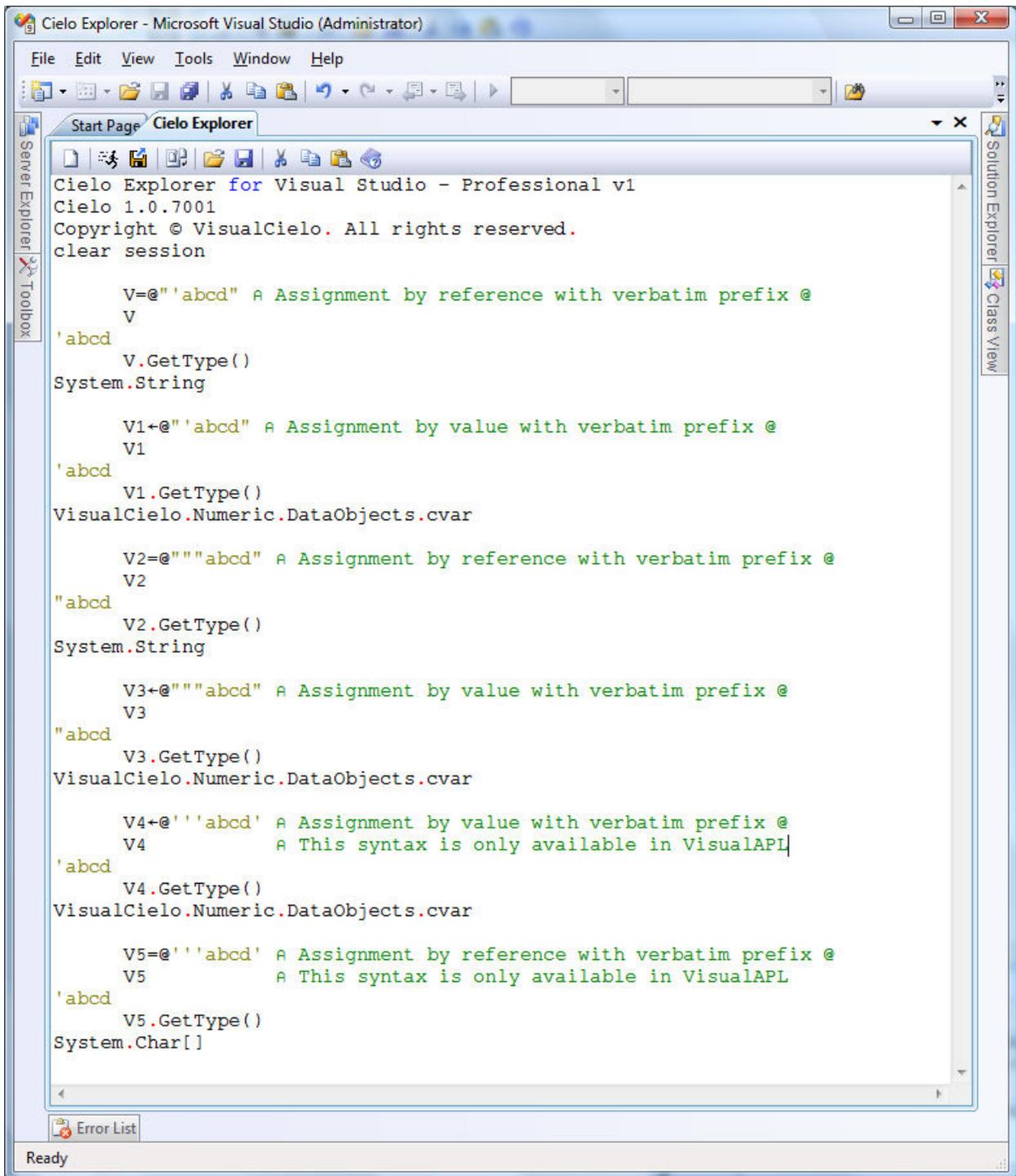
    T2="abcd" A Assignment by reference
    T2      A Syntax available only if VisualAPL
"abcd
    T2.GetType()
System.Char[]

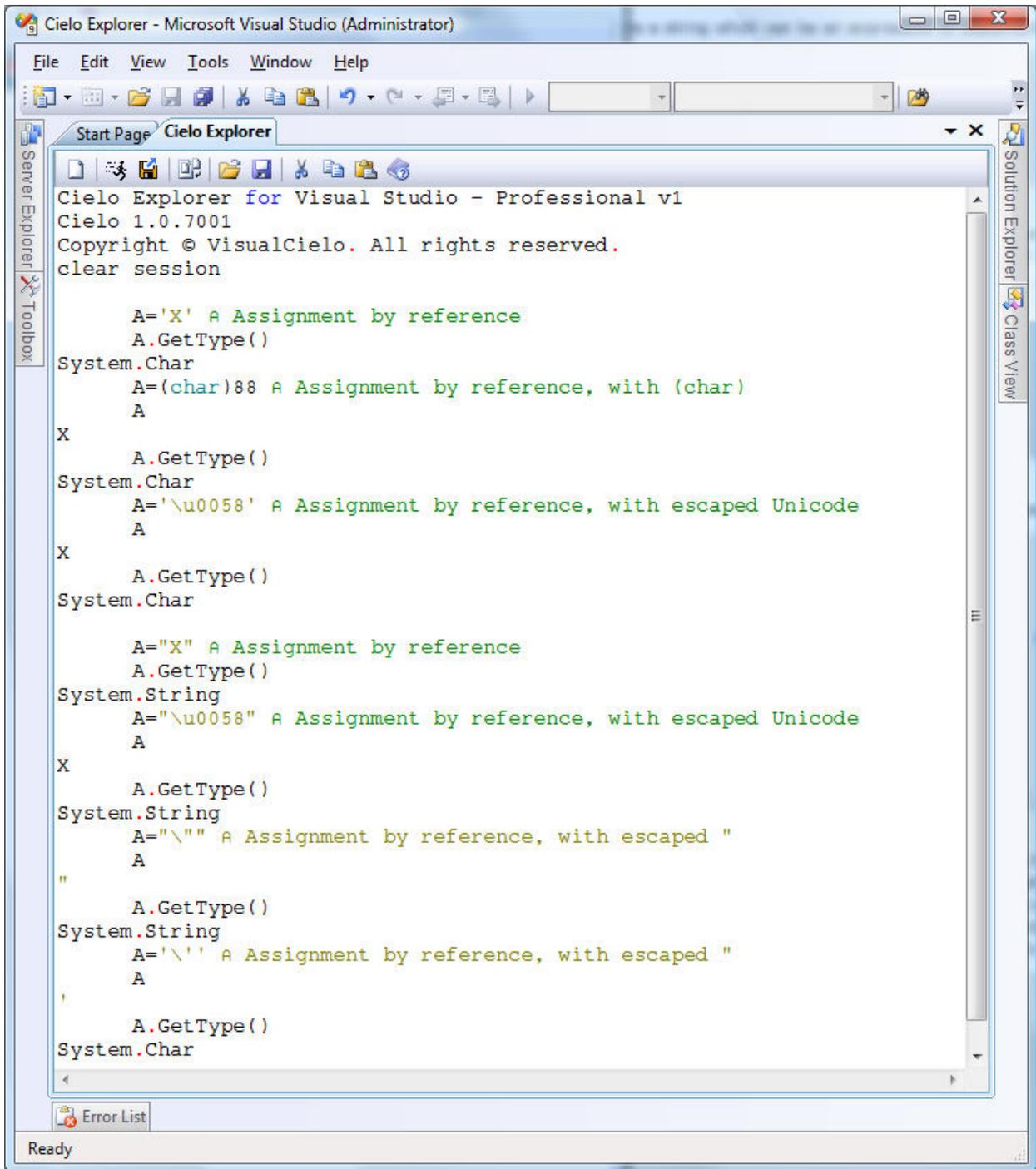
    T3<-"abcd" A Assignment by value
    T3      A Syntax available only if VisualAPI
"abcd
    T3.GetType()
VisualCielo.Numeric.DataObjects.cvar

    T5="\abcd" A Assignment by reference with escaped "
    T5
"abcd
    T5.GetType()
System.String

    T6<="\abcd" A Assignment by value with escaped "
    T6
"abcd
    T6.GetType()
VisualCielo.Numeric.DataObjects.cvar
```

Ready





The tutorial section of the "...\AplNext\VisualAPLProfessional\Documentation\Visual\_APL\_Help.chm" documentation file installed by VisualAPL provides a brief summary of string literals:

The screenshot shows a help window titled "Tutorial" with a navigation bar at the top containing "Hide", "Back", "Forward", and "Print" buttons. On the left is a "Contents" pane with a search box and a list of 38 topics. Topic 12, "12 Strings", is selected and highlighted in blue. The main content area on the right is titled "12 Strings" and contains the following text:

All strings or character arrays are unicode by definition. For compatibility with .Net, the backslash (\) is the escape character in string parsing.

For instance:

```
'hello'
hello
'doesn\t'
doesn't
"doesn't"
doesn't
"Hello", she said.'
"Hello", she said.
"\Hello\", she said."
"Hello", she said.
'Don\t', he asked?'
"Don't", he asked?
```

Implied line continuation occurs for string literals.

```
makestring {
    a - "this is a
line of text
over three lines"
    a - "this is a\n
line of text\n
over three lines"
```

For complete information on text literals, go to the C# language specification refer to the C# language specifications:

The screenshot shows an Internet Explorer browser window with the address bar displaying `http://msdn.microsoft.com/en-us/library/aa691090.aspx`. The page title is "2.4.4.5 String literals (C#)". The browser interface includes a menu bar (File, Edit, View, Favorites, Tools, Help), a search bar, and a navigation menu with options like Home, Library, Learn, Download, Support, Community, and Forums. The main content area is titled "C# Language Specification" and "2.4.4.5 String literals".

**C# Language Specification**  
**2.4.4.5 String literals**

C# supports two forms of string literals: regular string literals and verbatim string literals.

A regular string literal consists of zero or more characters enclosed in double quotes, as in "hello", and may include both simple escape sequences (such as \t for the tab character) and hexadecimal and Unicode escape sequences.

A verbatim string literal consists of an @ character followed by a double-quote character, zero or more characters, and a closing double-quote character. A simple example is @"hello". In a verbatim string literal, the characters between the delimiters are interpreted verbatim, the only exception being a *quote-escape-sequence*. In particular, simple escape sequences and hexadecimal and Unicode escape sequences are not processed in verbatim string literals. A verbatim string literal may span multiple lines.

*string-literal:*  
*regular-string-literal*  
*verbatim-string-literal*

*regular-string-literal:*  
" *regular-string-literal-characters*<sub>opt</sub> "

*regular-string-literal-characters:*  
*regular-string-literal-character*  
*regular-string-literal-characters* *regular-string-literal-character*

*regular-string-literal-character:*  
*single-regular-string-literal-character*  
*simple-escape-sequence*  
*hexadecimal-escape-sequence*  
*unicode-escape-sequence*

*single-regular-string-literal-character:*  
Any character except " (U+0022), \ (U+005C), and *new-line-character*

*verbatim-string-literal:*  
@" *verbatim-string-literal-characters*<sub>opt</sub> "

*verbatim-string-literal-characters:*  
*verbatim-string-literal-character*  
*verbatim-string-literal-characters* *verbatim-string-literal-character*

*verbatim-string-literal-character:*  
*single-verbatim-string-literal-character*  
*quote-escape-sequence*

*single-verbatim-string-literal-character:*  
Any character except "

*quote-escape-sequence:*  
"

A character that follows a backslash character (\) in a *regular-string-literal-character* must be one of the following characters: ', ", \, 0, a, b, f, n, r, t, u, v. Otherwise, a compile-time error occurs.

2.4.4.4 Character literals (C#) - Internet Explorer provided by Dell  
 http://msdn.microsoft.com/en-us/library/aa691087(VS.71).aspx

File Edit View Favorites Tools Help  
 Google C# char[] literals Go Bookmarks 23 blocked Check Settings

Visual Studio Developer Center  
 Home Library Learn Download Support Community Forums

Printer Friendly Version Add To Favorites Send Click to Rate and Give Feedback

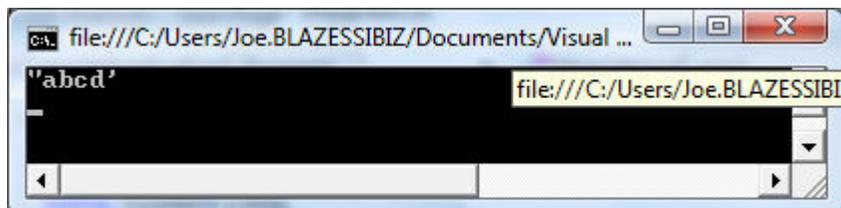
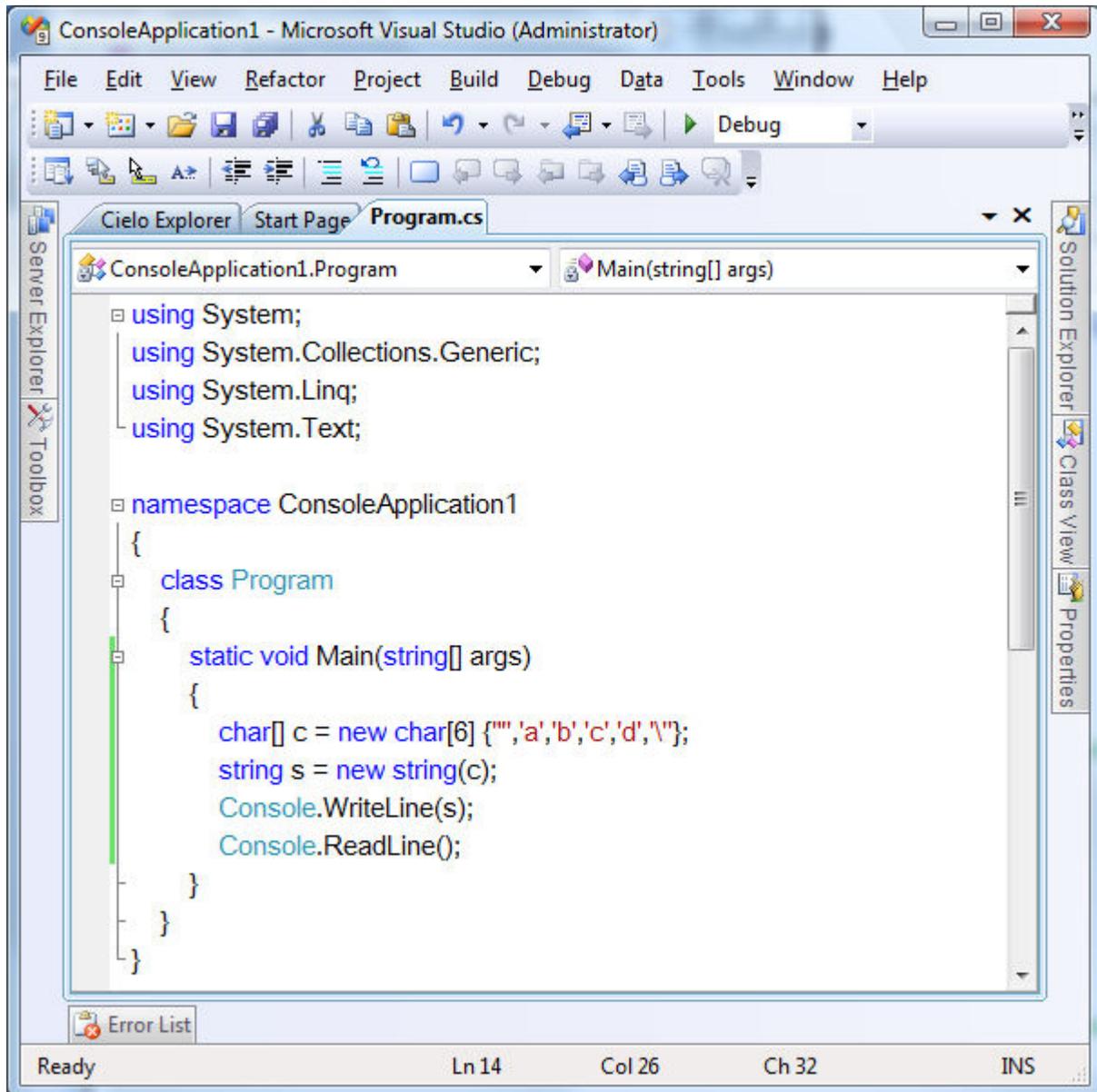
Visual C# Language  
 C# Language Tour  
 C# Compiler Options  
 Visual C# Code Wizards  
 C# Language Specification  
 1. Introduction  
 2. Lexical structure  
 2.1 Programs  
 2.2 Grammars  
 2.3 Lexical analysis  
 2.4 Tokens  
 2.4.1 Unicode character escape sequences  
 2.4.2 Identifiers  
 2.4.3 Keywords  
 2.4.4 Literals  
 2.4.4.1 Boolean literals  
 2.4.4.2 Integer literals  
 2.4.4.3 Real literals  
 2.4.4.4 Character literals  
 2.4.4.5 String literals  
 2.4.4.6 The null literal  
 2.4.5 Operators and punctuators  
 2.5 Pre-processing directives  
 3. Basic concepts  
 4. Types  
 5. Variables  
 6. Conversions  
 7. Expressions  
 8. Statements  
 9. Namespaces  
 10. Classes  
 11. Structs  
 12. Arrays  
 13. Interfaces  
 14. Enums

C# Language Specification  
**2.4.4.4 Character literals**  
 A character literal represents a single character, and usually consists of a character in quotes, as in 'a'.  
*character-literal:*  
`' character '`  
*character:*  
`single-character  
 simple-escape-sequence  
 hexadecimal-escape-sequence  
 unicode-escape-sequence`  
*single-character:*  
 Any character except ' (U+0027), \ (U+005C), and *new-line-character*  
*simple-escape-sequence:* one of  
`' \ \0 \a \b \f \n \r \t \v`  
*hexadecimal-escape-sequence:*  
`\x hex-digit hex-digitopt hex-digitopt hex-digitopt`  
 A character that follows a backslash character (\) in a *character* must be one of the following characters: ', ", \, 0, a, b, f, n, r, t, u, v, x. Otherwise, a compile-time error occurs.  
 A hexadecimal escape sequence represents a single Unicode character, with the value formed by the hexadecimal number following "\x".  
 If the value represented by a character literal is greater than U+FFFF, a compile-time error occurs.  
 A Unicode character escape sequence (Section 2.4.1) in a character literal must be in the range U+0000 to U+FFFF.  
 A simple escape sequence represents a Unicode character encoding, as described in the table below.

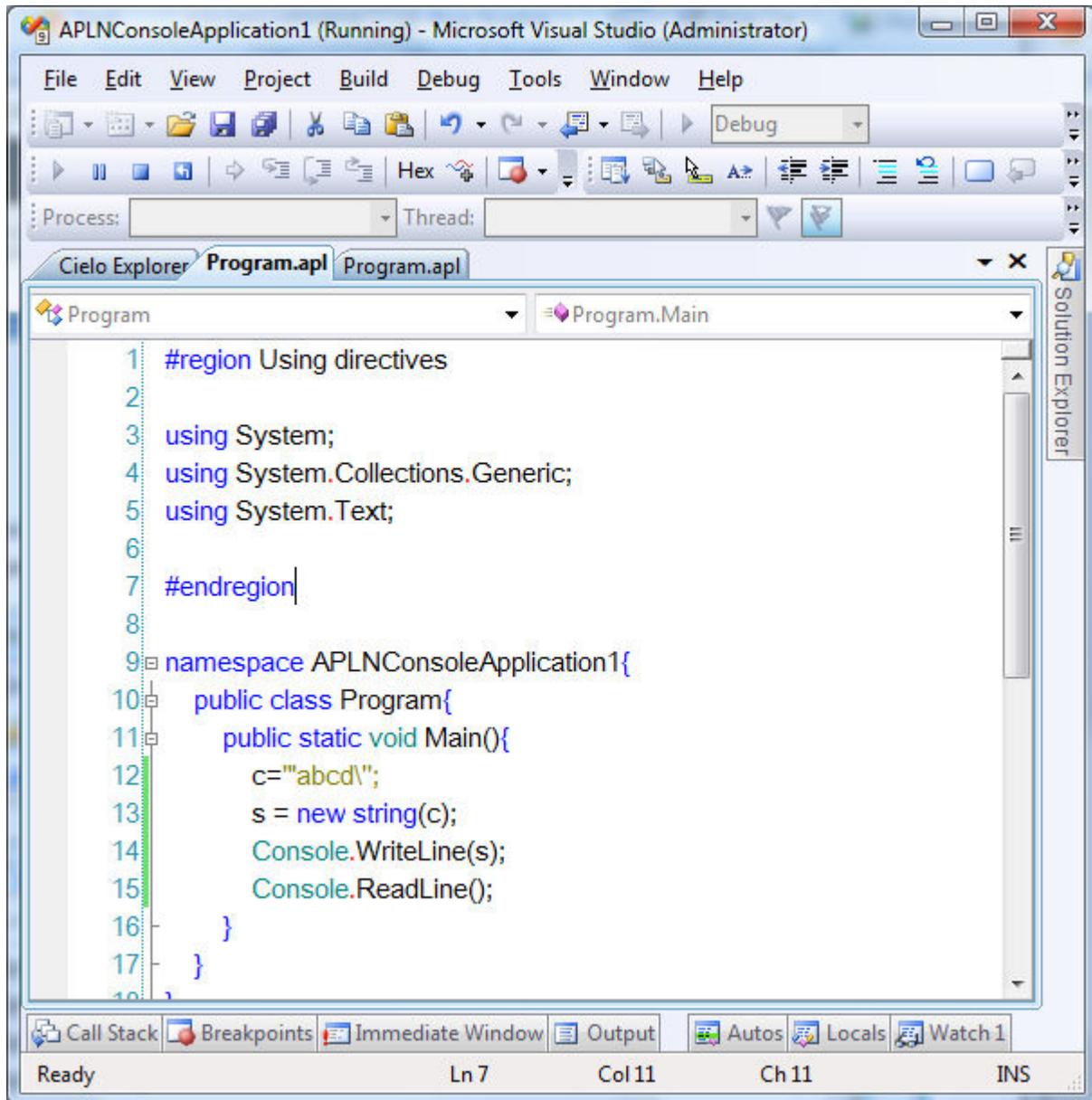
Escape sequence	Character name	Unicode encoding
<code>'</code>	Single quote	<code>0x0027</code>
<code>"</code>	Double quote	<code>0x0022</code>
<code>\</code>	Backslash	

Internet | Protected Mode: Off 100%

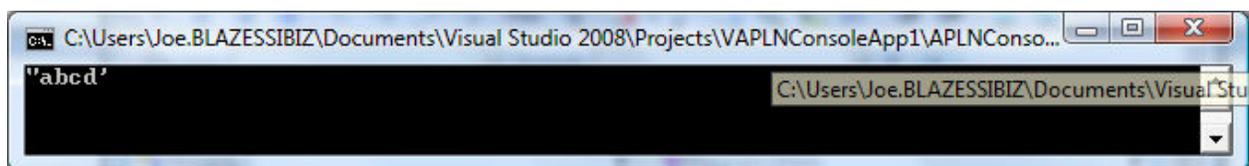
Here is a C# console project which displays char[] data to the screen. Notice how only one character at a time may be specified in C#. Also note the escaped ' in the 6<sup>th</sup> element of the char[] array. To properly display the char[] array value using the Console.WriteLine() method, a new string is created from the char[] array.



Here is the analogous VisualAPL console project. Notice how the char[] '...' assignment by reference syntax has been extended in VisualAPL to permit the specification of more than one character in the '...' format of the assignment. The C# char[6] {'"', 'a', 'b', 'c', 'd', '\\'} syntax could have been used instead of the VisualAPL syntax "'abcd\\", but the latter syntax is certainly more convenient.

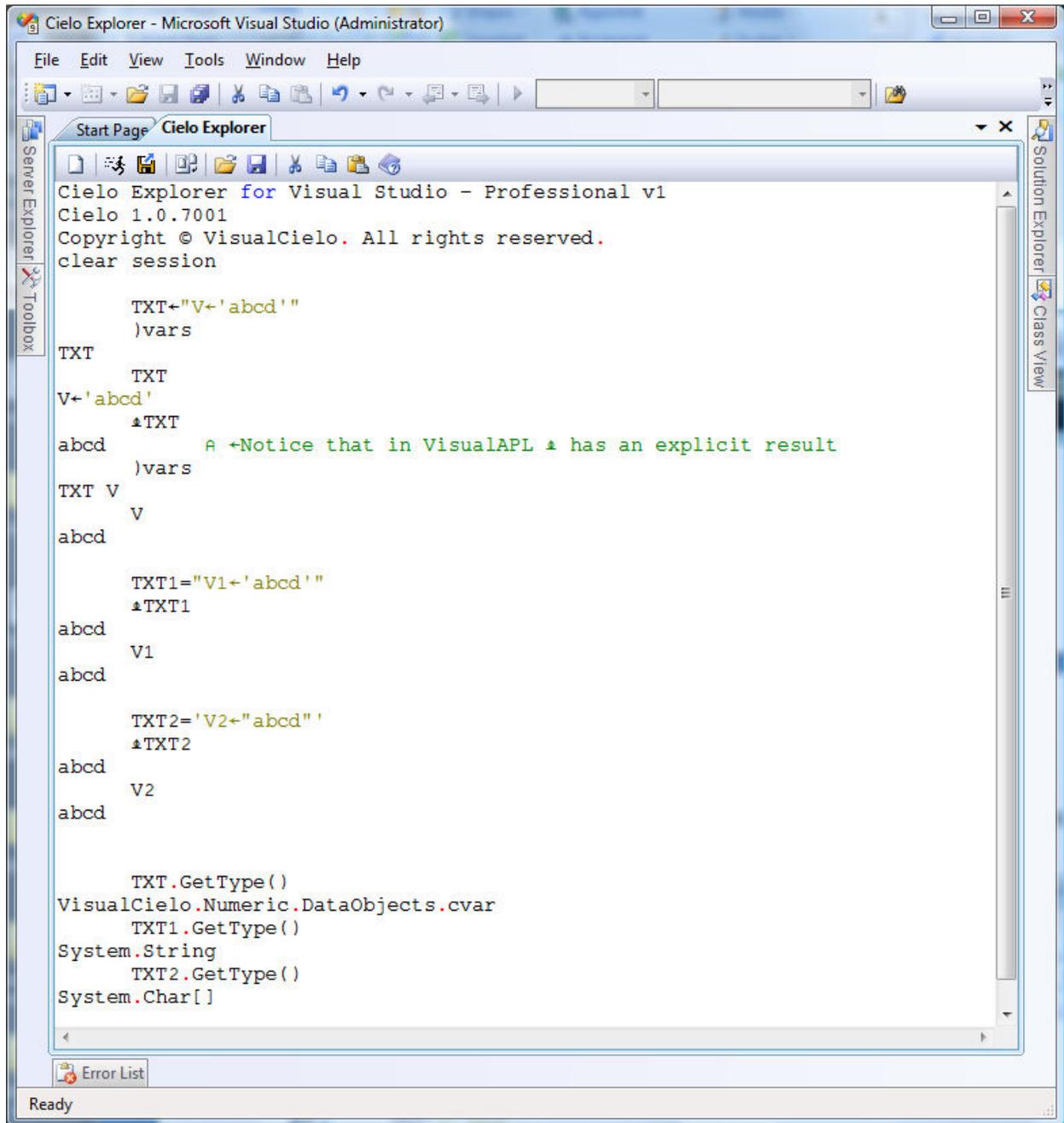


```
1 #region Using directives
2
3 using System;
4 using System.Collections.Generic;
5 using System.Text;
6
7 #endregion
8
9 namespace APLNConsoleApplication1{
10     public class Program{
11         public static void Main(){
12             c="abcd";
13             s = new string(c);
14             Console.WriteLine(s);
15             Console.ReadLine();
16         }
17     }
18 }
```



```
C:\Users\Joe.BLAZESSIBIZ\Documents\Visual Studio 2008\Projects\VAPLNConsoleApp1\APLNConso...
"abcd"
```

The VisualAPL primitive execute operation can be applied to text variables with type cvar (Cielo Variable), string or char[]:



```
Cielo Explorer for Visual Studio - Professional v1
Cielo 1.0.7001
Copyright © VisualCielo. All rights reserved.
clear session

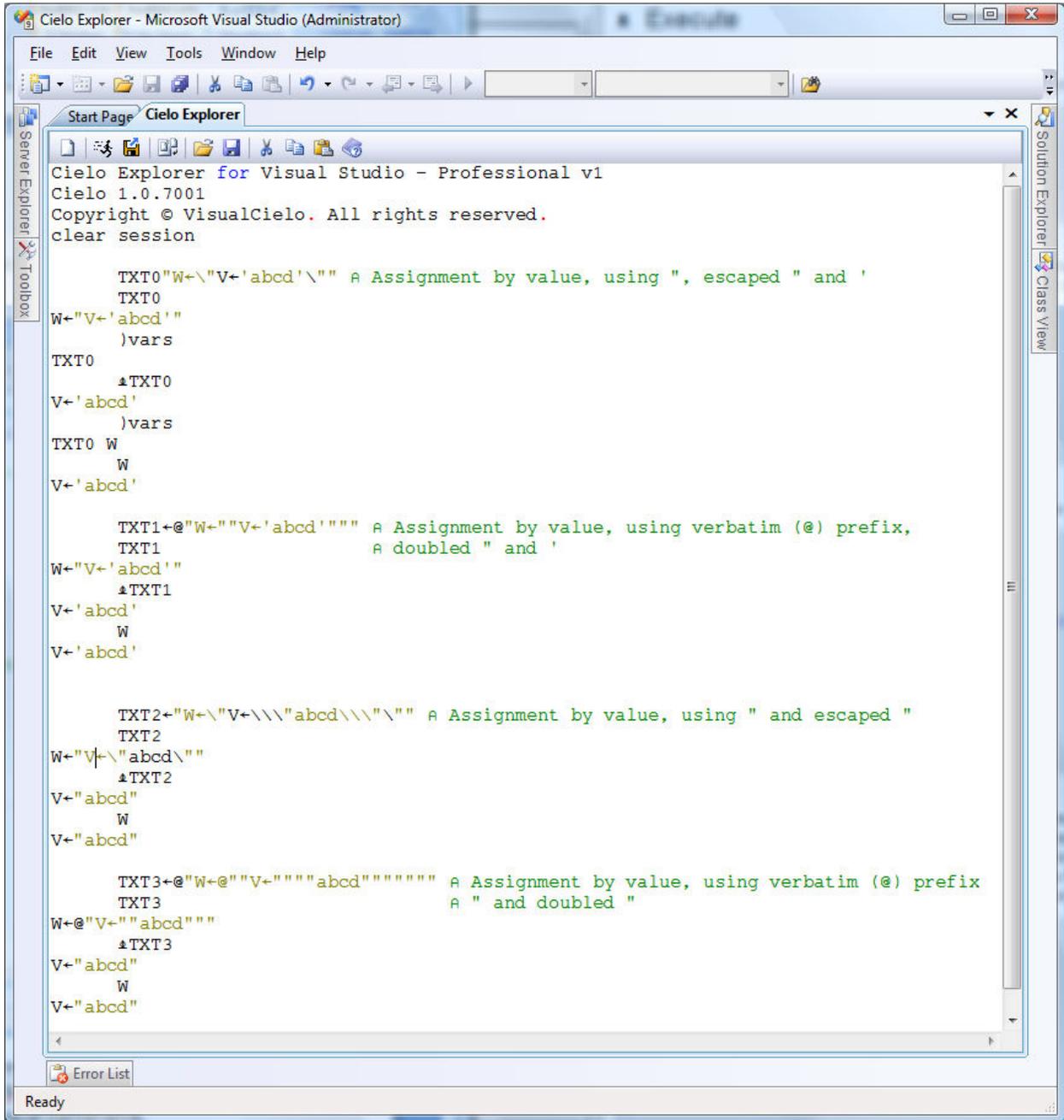
    TXT←"V+'abcd'"
    )vars
TXT
    TXT
V←'abcd'
    ▲TXT
abcd      A ←Notice that in VisualAPL ▲ has an explicit result
    )vars
TXT V
    V
abcd

    TXT1="V1+'abcd'"
    ▲TXT1
abcd
    V1
abcd

    TXT2='V2+"abcd"'
    ▲TXT2
abcd
    V2
abcd

    TXT.GetType()
VisualCielo.Numeric.DataObjects.cvar
    TXT1.GetType()
System.String
    TXT2.GetType()
System.Char[]
```

Using the VisualAPL primitive execute on a text literal which contains a deeper level of embedded quotes:



```
Cielo Explorer for Visual Studio - Professional v1
Cielo 1.0.7001
Copyright © VisualCielo. All rights reserved.
clear session

    TXT0"W+\V+'abcd\'\" A Assignment by value, using ", escaped " and '
    TXT0
W+"V+'abcd'"
)vars
TXT0
    ▲TXT0
V+'abcd'
)vars
TXT0 W
    W
V+'abcd'

    TXT1+@"W+"V+'abcd'"" A Assignment by value, using verbatim (@) prefix,
    TXT1                A doubled " and '
W+"V+'abcd'"
    ▲TXT1
V+'abcd'
    W
V+'abcd'

    TXT2+"W+\V+\\\\"abcd\\\\"\" A Assignment by value, using " and escaped "
    TXT2
W+"V+\\"abcd\'\"
    ▲TXT2
V+"abcd"
    W
V+"abcd"

    TXT3+@"W+@"V+""abcd"""" A Assignment by value, using verbatim (@) prefix
    TXT3                A " and doubled "
W+@"V+""abcd""""
    ▲TXT3
V+"abcd"
    W
V+"abcd"
```

Note that a user-defined VisualAPL function (using the operator or method function signature) in a VisualAPL class library may be a better solution than using the VisualAPL primitive execute operator on a text literal variable considering processing performance and ease of maintenance.