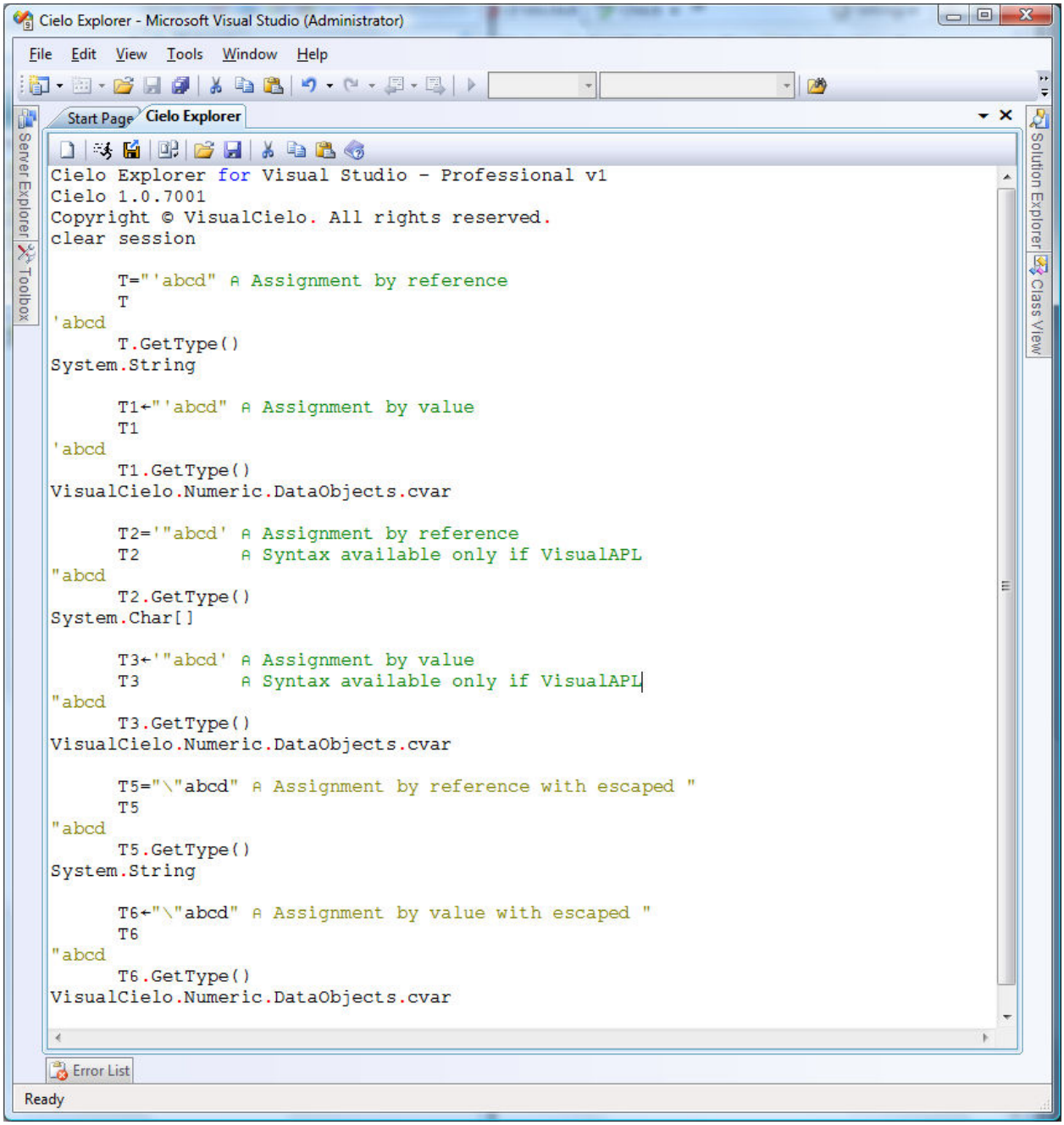
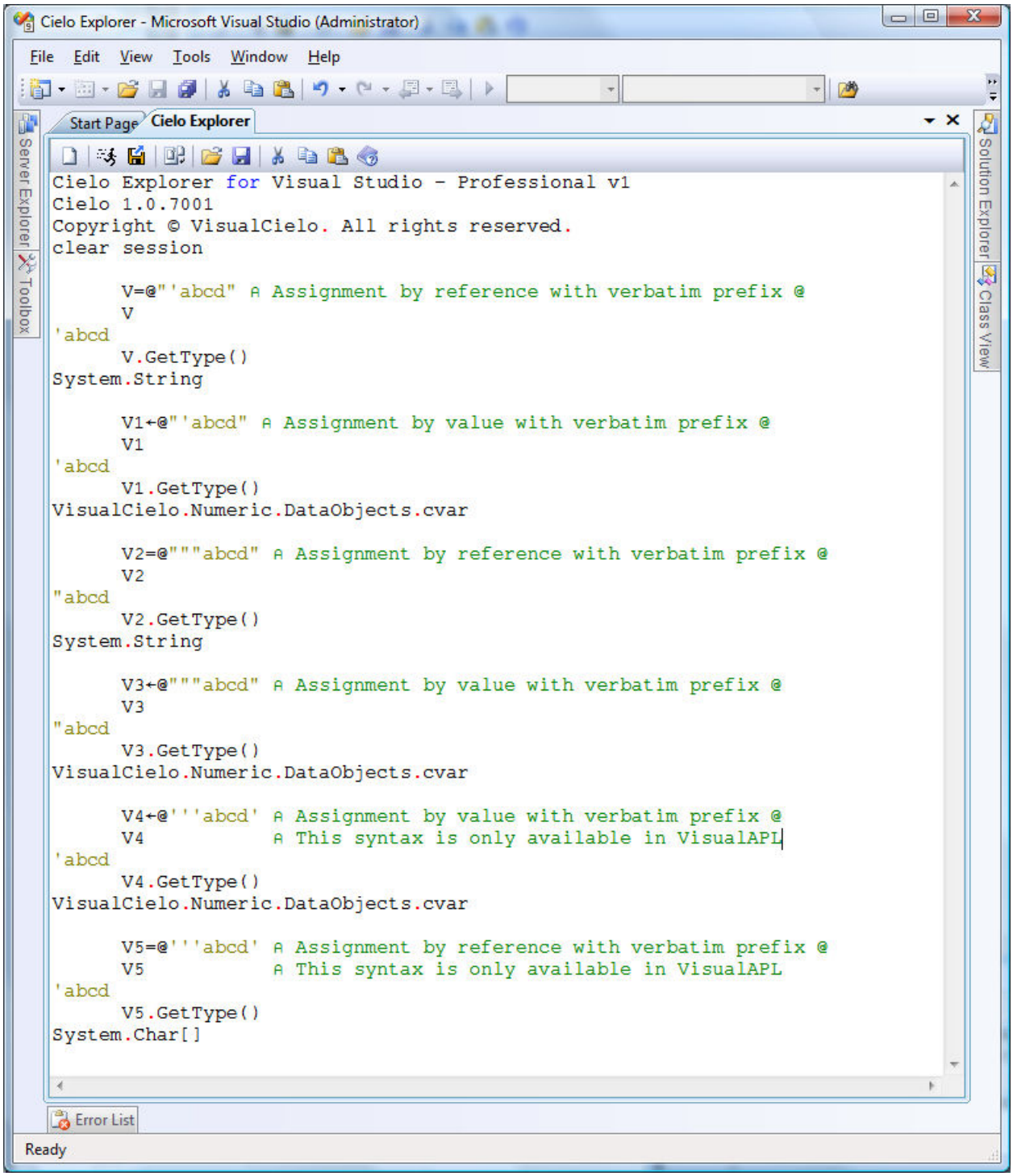
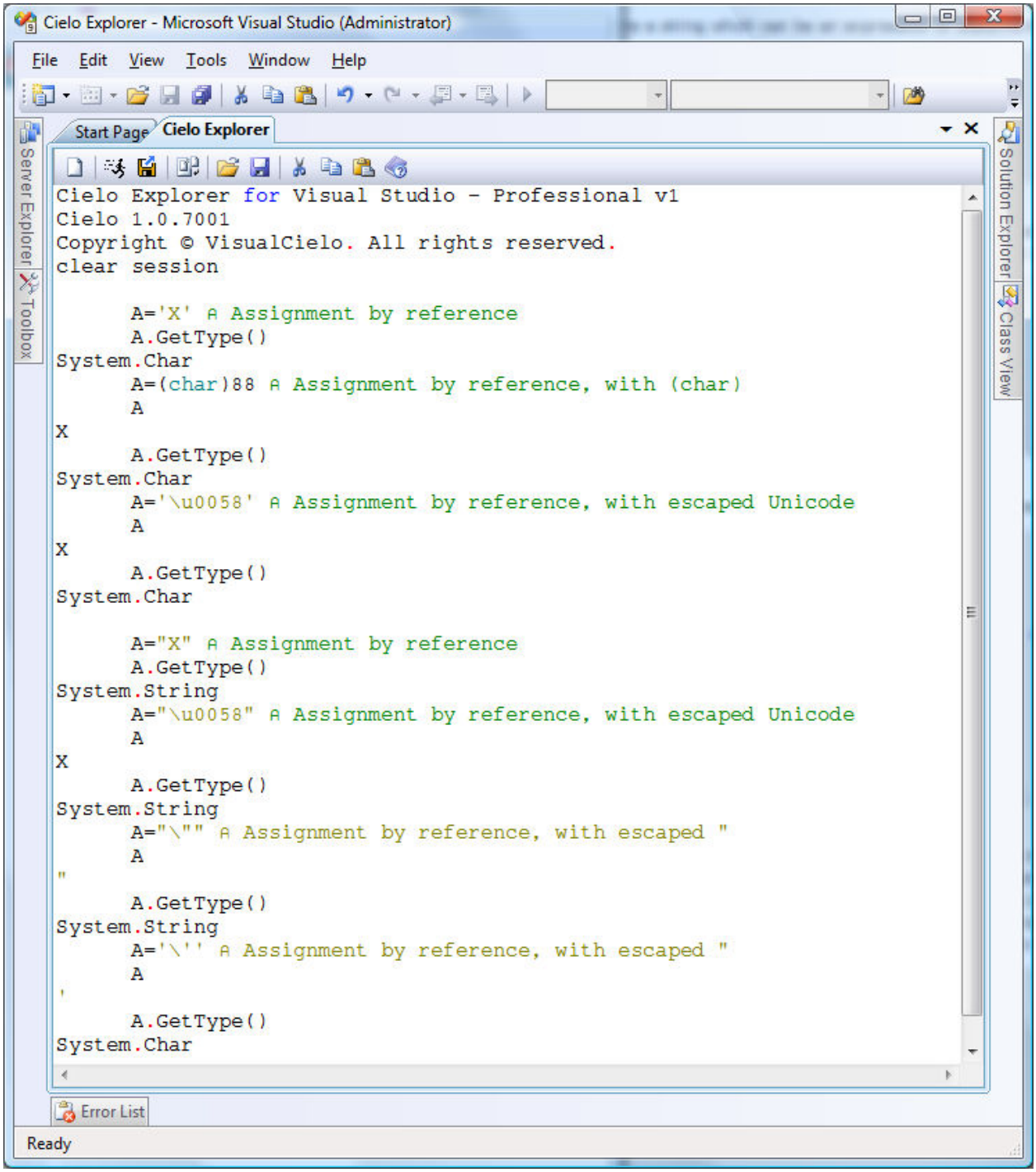
VisualAPL Text Literals

- 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[].

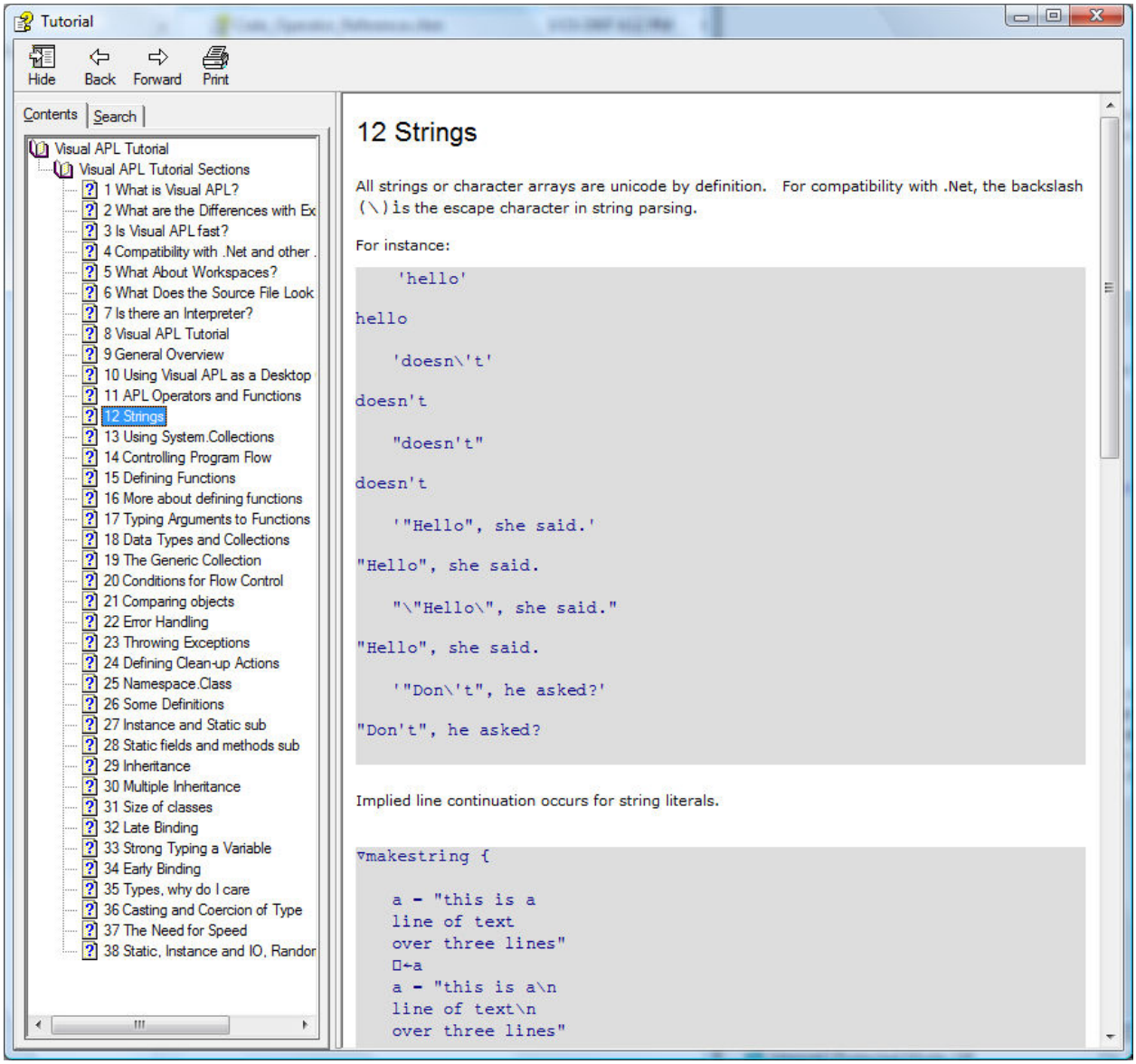
Study the following examples closely and try them out in the Cielo Explorer:



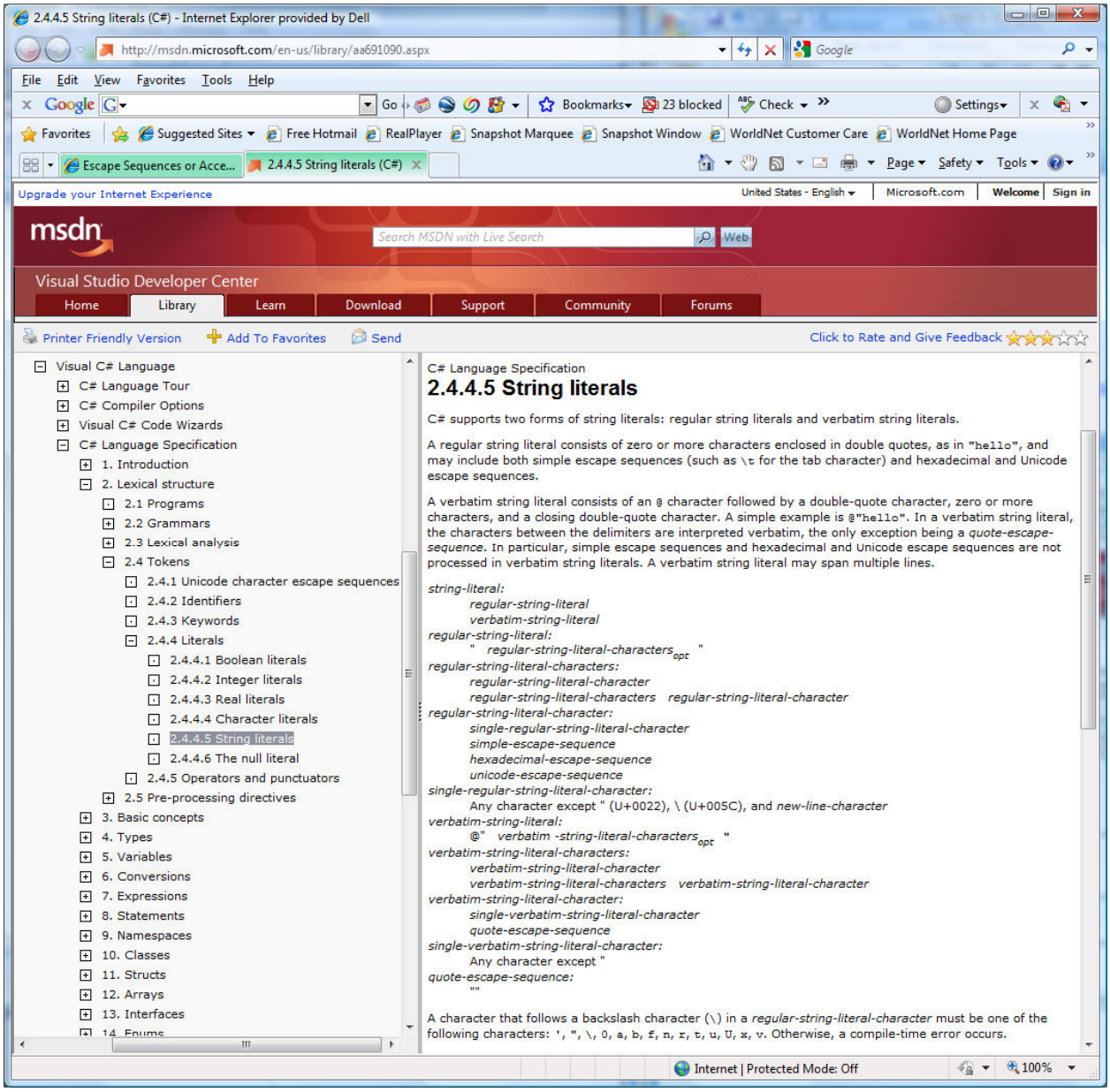

Cielo Explorer - Microsoft Visual Studio (Administrator)
File Edit View Tools Window Help
Start Page
Cielo Explorer
Server Explorer
Toolbox
Solution Explorer
Class View
Cielo Explorer for Visual Studio - Professional v1
Cielo 1.0.7001
Copyright © VisualCielo. All rights reserved.
clear session
V←@"'abcd" ⍝ Assignment by reference with verbatim prefix @
V
'abcd
V.GetType()
System.String
V1←@"'abcd" ⍝ Assignment by value with verbatim prefix @
V1
'abcd
V1.GetType()
VisualCielo.Numeric.DataObjects.cvar
V2←@"""abcd" ⍝ Assignment by reference with verbatim prefix @
V2
"abcd
V2.GetType()
System.String
V3←@"""abcd" ⍝ Assignment by value with verbatim prefix @
V3
"abcd
V3.GetType()
VisualCielo.Numeric.DataObjects.cvar
V4←@'''abcd' ⍝ Assignment by value with verbatim prefix @
V4 ⍝ This syntax is only available in VisualAPL
'abcd
V4.GetType()
VisualCielo.Numeric.DataObjects.cvar
V5←@'''abcd' ⍝ Assignment by reference with verbatim prefix @
V5 ⍝ This syntax is only available in VisualAPL
'abcd
V5.GetType()
System.Char[]
Error List
Ready

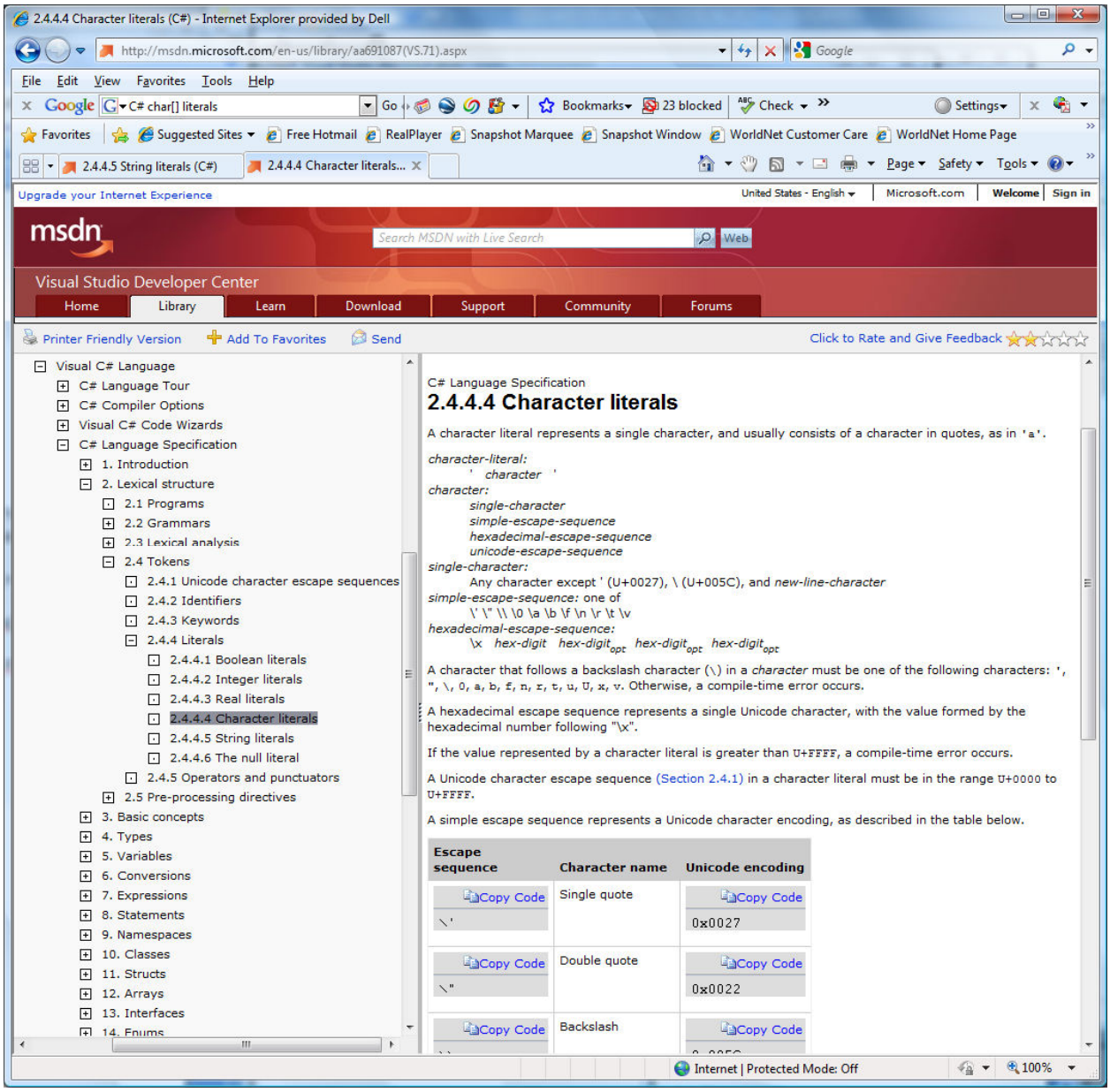

Cielo Explorer - Microsoft Visual Studio (Administrator)
File Edit View Tools Window Help
Start Page
Cielo Explorer
Server Explorer
Toolbox
Solution Explorer
Class View
Cielo Explorer for Visual Studio - Professional v1
Cielo 1.0.7001
Copyright © VisualCielo. All rights reserved.
clear session
A='X' # Assignment by reference
A.GetType()
System.Char
A=(char)88 # Assignment by reference, with (char)
A
X
A.GetType()
System.Char
A='\u0058' # Assignment by reference, with escaped Unicode
A
X
A.GetType()
System.Char
A="X" # Assignment by reference
A.GetType()
System.String
A="\u0058" # Assignment by reference, with escaped Unicode
A
X
A.GetType()
System.String
A="\"" # Assignment by reference, with escaped "
A
"
A.GetType()
System.String
A='\'' # Assignment by reference, with escaped '
A
'
A.GetType()
System.Char
Error List
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:

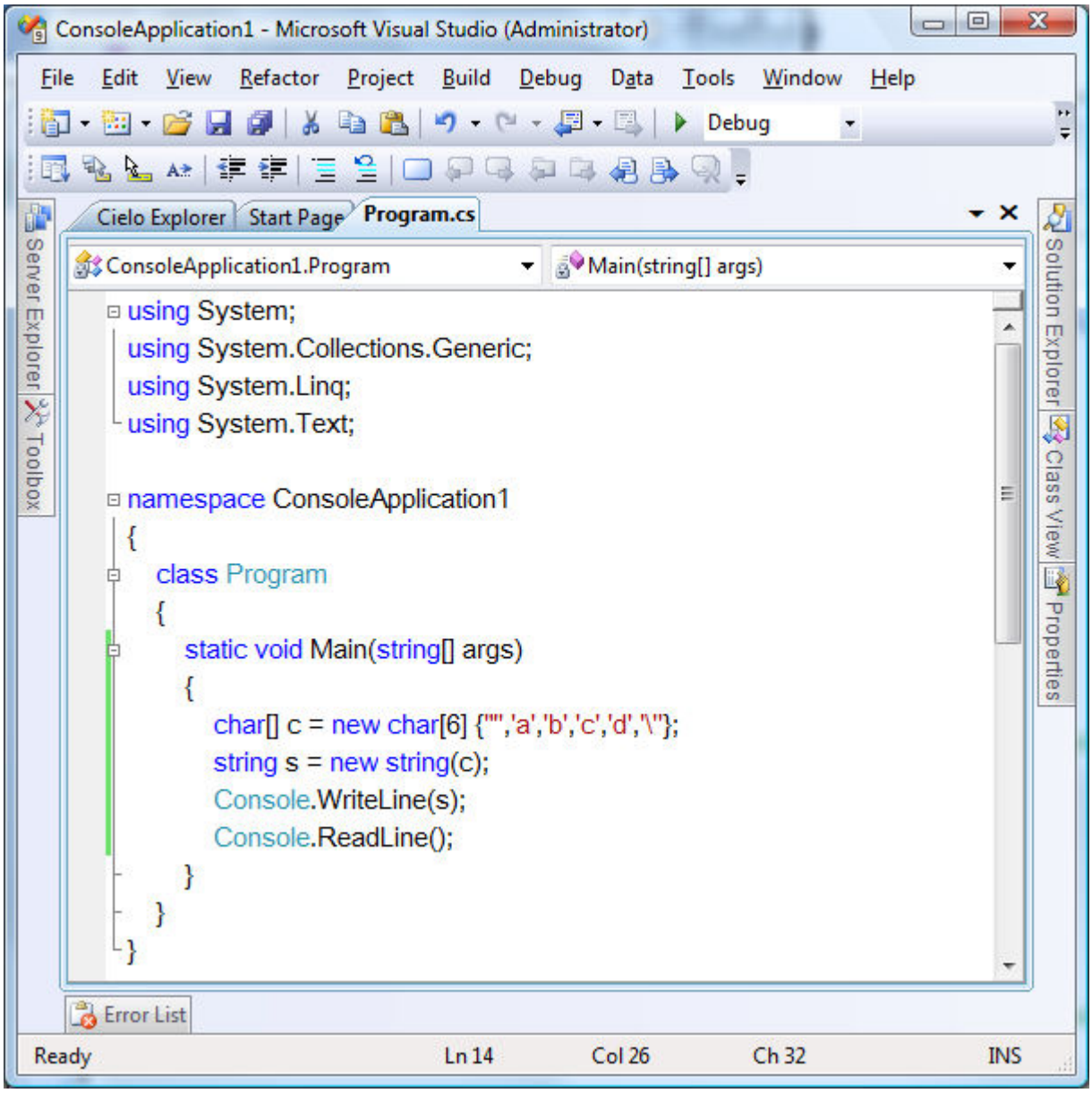


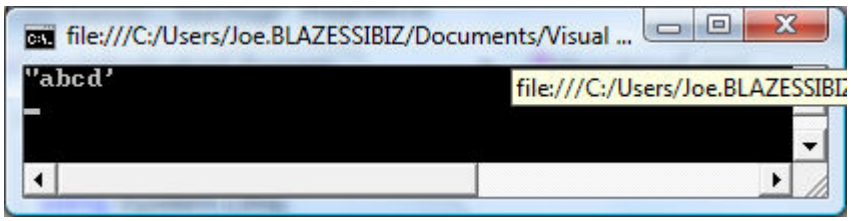
For complete information on text literals, go to the C# language specification refer to the C# language specifications:



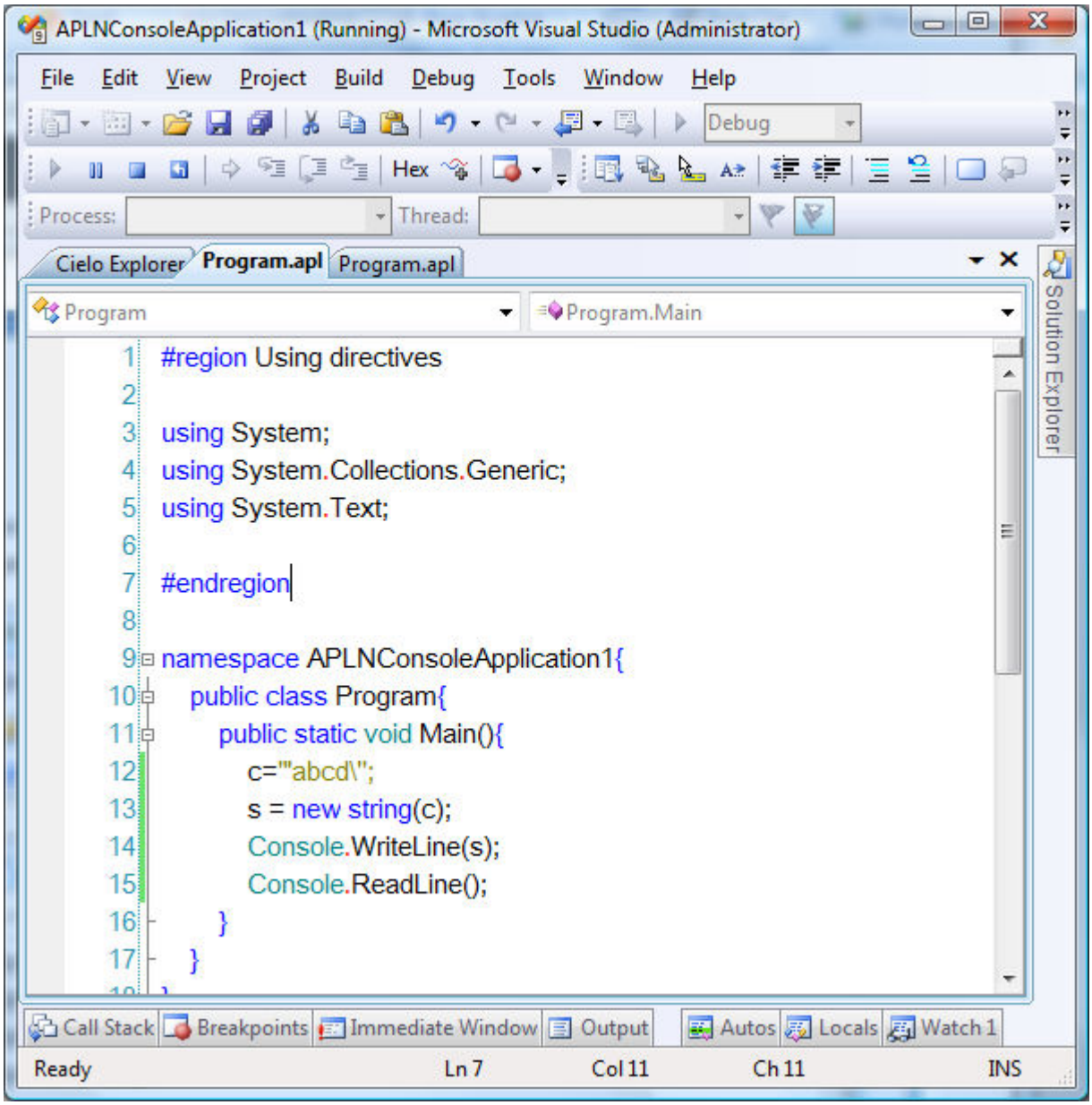

2.4.4.4 Character literals (C#) - Internet Explorer provided by Dell
http://msdn.microsoft.com/en-us/library/aa691087(VS.71).aspx
File Edit View Favorites Tools Help
C# char[] literals
Favorites
2.4.4.5 String literals (C#)
2.4.4.4 Character literals...
Upgrade your Internet Experience
United States - English
Microsoft.com
Welcome
Sign in
msdn
Search MSDN with Live Search
Web
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, U, x, v. 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
Copy Code
\'
Single quote
Copy Code
0x0027
Copy Code
\"
Double quote
Copy Code
0x0022
Copy Code
Backslash
Copy Code
Internet | Protected Mode: Off
100%

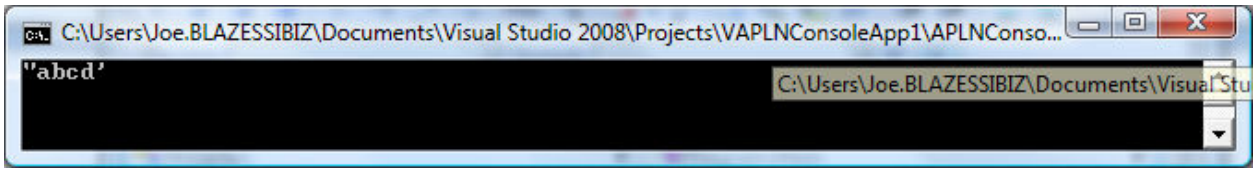
Here is a C# console project which displays char[] data to the screen. Notice how only one character at a time may be specified in C#. Also note the escaped ‘ in the 6th element of the char[] array. To properly display the char[] array value using the Console.WriteLine() method, a new string is created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



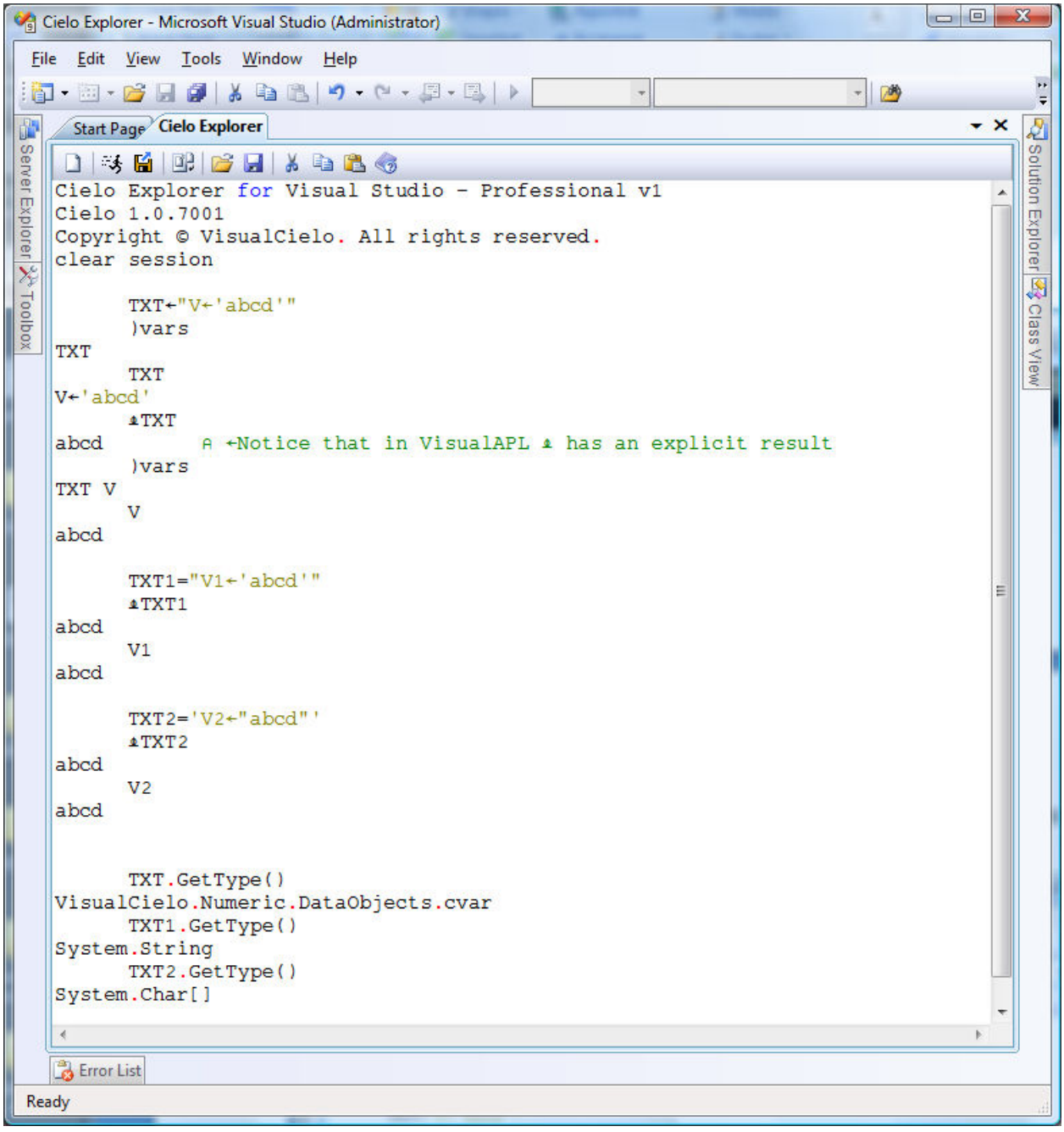


Here is the analogous VisualAPL console project. Notice how the char[] '…' assignment by reference syntax has been extended in VisualAPL to permit the specification of more than one character in the '…' format of the assignment. The C# char[6] {'"','a','b','c','d','\'} syntax could have been used instead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

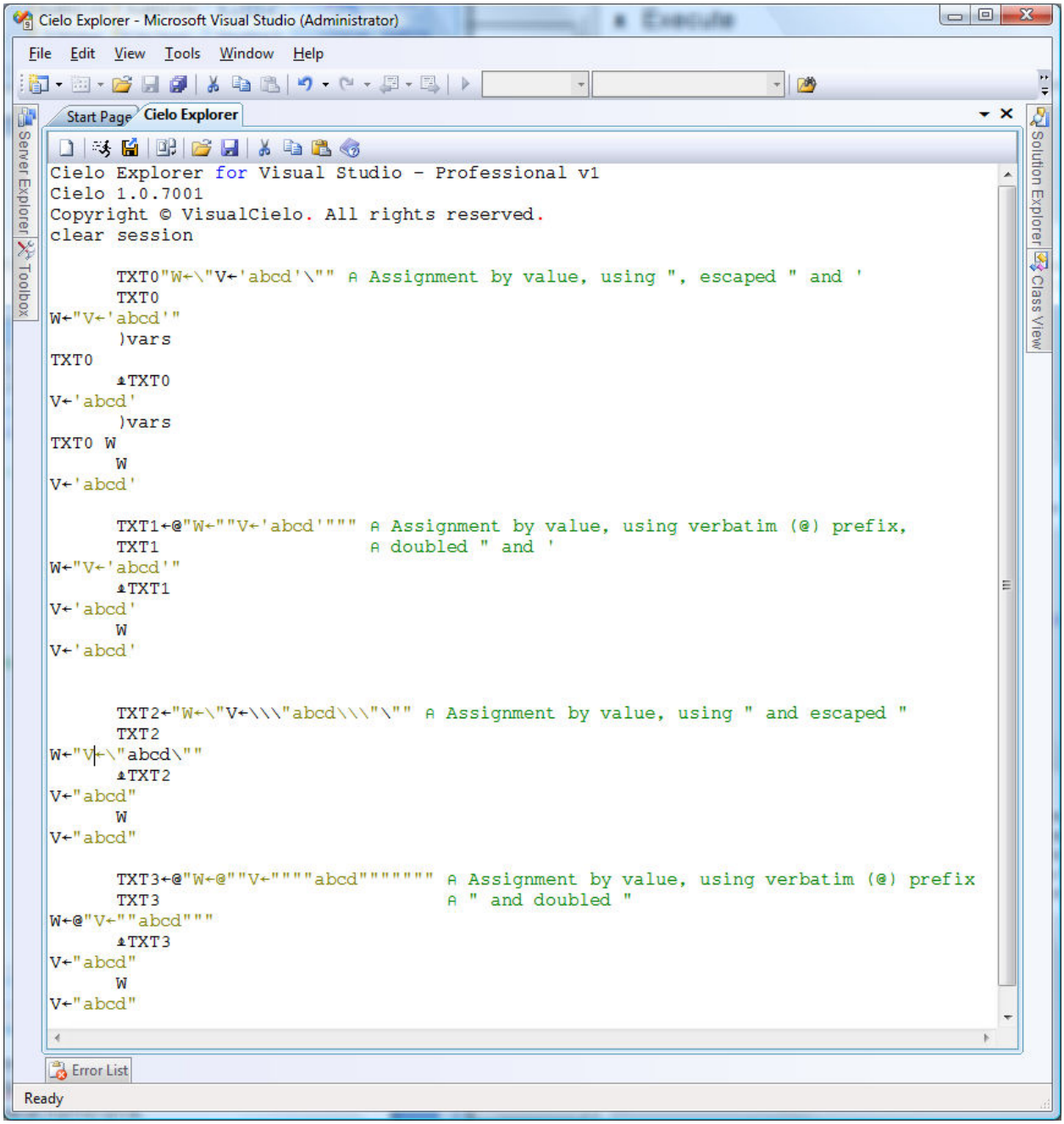




The VisualAPL primitive execute operation can be applied to text variables with type cvar (Cielo Variable), string or char[]:



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



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.