See Also

http://en.wikipedia.org/wiki/Unicode

http://en.wikipedia.org/wiki/UTF-16

http://en.wikipedia.org/wiki/UTF-8

Embedded Nulls in Literal Strings

A null in a byte-oriented string is a byte with the value 0.

It is possible to embed an null byte in a string:

```
String test = "A\x00B"                        // OK in Igor Pro 7 or later
Print strlen(text)                                       // Prints 3
Print char2num(test[0]),char2num(test[1]),char2num(test[2]) // Prints 65 0 66
```

Here Igor converts the escape sequence \x00 to a null byte.

You typically have no need to embed a null in an Igor string because strings are usually used to store readable text and null does not represent a readable character. The need might arise, however, if you are using the string to store binary rather than text data. For example, if you need to send a small amount of binary data to an instrument, you can do so using \x escape sequences to represent the data in a literal string.

Although Igor allows you to embed nulls in literal strings, other parts of Igor are not prepared to handle them. For example:

```
Print test           // Prints "A", not "A<null>B"
```

In C null is taken to mean "end-of-string". Because of the use of C strings and C library routines in Igor, many parts of Igor will treat an embedded null as end-of-string. That is why the Print statement above prints just "A".

The bottom line is: You can store binary data, including nulls, in an Igor string but many parts of Igor that expect readable text will treat a null as end-of-string. See **Working With Binary String Data** for further discussion.

String Indexing

Indexing can extract a part of a string. This is done using a string expression followed by one or two numbers in brackets. The numbers are byte positions. Zero is the byte position of the first byte; n-1 is the byte position of the last byte of an n byte string value. For example, assume we create a string variable called s1 and assign a value to it as follows:

```
String s1="hello there"
```

h	e	l	l	o		t	h	e	r	e
0	1	2	3	4	5	6	7	8	9	10

Then,

```
Print s1[0,4]              prints   hello
Print s1[0,0]              prints   h
Print s1[0]                prints   h
Print s1[1]+s1[2]+s1[3]    prints   ell
Print (s1+" jack")[6,15]   prints   there jack
```

A string indexed with one index, such as s1[p], is a string with one byte in it if p is in range (i.e. 0 ≤ p ≤ n-1). s1[p] is a string with no bytes in it if p is not in range. For example:

```
Print s1[0]                prints   h
Print s1[-1]               prints   (nothing)
Print s1[10]               prints   e
```