

uCalc String Library

uCalc Strings Library consists of a set of string-handling routines whose names will seem very familiar, especially to those who use either .NET's Microsoft.VisualBasic.Strings module, VB 6, or PowerBASIC, but with support for uCalc Patterns, which infuses each routine with advanced parsing capabilities.

If you are familiar with such routines as InStr, Mid(), Left(), Right(), etc, you already know most of what you need to know to get started. This documentation focuses mainly on the additional functionality that uCalc brings to the table. The general description of each routine is similar to what you may be used to. The main differences are that the MatchString argument accepts uCalc Patterns (instead of just plain strings); and numeric arguments can be measured either in terms of characters, tokens, blocks, or expressions; and all routines have an additional Options parameter, which lets you select such things as whether the function call should be whitespace-sensitive and case-sensitive. By default case-sensitivity and white-spaces sensitivity is off, and tokens (not characters) are the unit of measurement, except where indicated otherwise. Some routines might be more familiar particularly to PowerBASIC users.

Familiar routines are listed towards the top, and uCalc-specific items are explained in further detail towards the bottom. As is common in BASIC dialects, character positions start at 1 (not 0). Functions towards the top may have a greater variety of examples. In general, some of the same features in those earlier examples apply to other functions further down. Some of the later topics may introduce examples that are unique to either the given function or similar functions. In the examples the line of text below the library command represents the content of the MainString variable.

For details on uCalc patterns, visit <http://www.ucalc.com/docs/Patterns.htm>

ucInStr (*Start*, *MainString*, *MatchString* [, *Options*])

Returns the location of *MatchString* within *MainString*.

Start - Starting character position from where to begin the search

MainString - String within which to search for *MatchString*

MatchString - String or pattern to find within *MainString*

Example: InStr with default setting

```
ucInStr(1, MainString, "a test")
```

```
THIS IS A TEST, "This is a test", (a test), a test, a test.
```

```
    ^  
    Returns 9
```

Explanation: With the default setting, "a test" (third argument) is a pattern and not a literal string. This pattern consists of the tokens "a" followed by "test". Spacing between the two does not matter, nor does casing (upper case/lower case).

Example: InStr with special patterns

```
uInStr(1, MainString, "{whatever}")  
THIS IS A TEST, "This is a test", (a test), a test, a test.  
^  
Returns 37
```

```
uInStr(1, MainString, "is {etc:2}")  
THIS IS A TEST, "This is a test", (a test), a test, a test.  
^  
Returns 6
```

Explanation: In the first part, it matches the first pair of parenthesis with some text inside. The second one matches the token "is" followed by the next two tokens. See <http://www.ucalc.com/docs/Patterns.htm> for more details on uCalc Patterns.

Example: InStr with ucChar property

```
uInStr(1, MainString, "a test", ucChar)  
THIS IS A TEST, "This is a test", (a test), a test, a test.  
^  
Returns 28
```

Explanation: **ucChar** makes it compare character by character (the way traditional VB's InStr() function would do it), instead of uCalc's default token by token search. This is similar to not using uCalc patterns.

Example: InStr with ucCase property

```
uInStr(1, MainString, "a test", ucCase)  
THIS IS A TEST, "This is a test", (a test), a test, a test.  
^  
Returns 38
```

Explanation: Here we introduce **ucCase**, which makes the search case sensitive. It skips over "A TEST", since it is all caps. Note that it also skips over the second literal occurrence, because it is within quotes. By default comparisons are done token by token, and "This is a test" (including surrounding quotes) represents one token, so it does not match the two separate tokens "a" followed by "test". Likewise, `uInStr(1, MainString, "is")` would match the "IS" token following "THIS", and not the "IS" that occurs within the word "THIS".

Example: InStr with ucSpace property

```
uInStr(1, MainString, "a test", ucSpace)  
THIS IS A TEST, "This is a test", (a test), a test, a test.
```

Explanation: Here the pattern is whitespace-sensitive. The given pattern has exactly one space, so it skips over occurrences where there is more than one space between "a" and "test".

Example: InStr with ucBlock property

ucInStr(1, MainString, "a test", **ucBlock**)

```
THIS IS A TEST, "This is a test", (a test), a test, a test.  
                                     ^  
                                     Returns 48
```

ucInStr(1, MainString, "(a test)", **ucBlock**)

```
THIS IS A TEST, "This is a test", (a test), a test, a test.  
                                     ^  
                                     Returns 37
```

Explanation: So far, we've searched character by character, or token by token. Here, we are searching block by block. A block consists of either one token, or if the token was defined with the block property, the block starts with that token and ends with the closing token associated with it. By default the following block tokens pairs are defined (and), { and }, and (and). The first part of the example skips over "(a test)", and matches the following occurrence because (a test), including the parenthesis, represents one block unit, and it is not equal to "a test". In the second example "(a test)" matches the given pattern "(a test)".

Example: InStr with combined properties

ucInStr(1, MainString, "testing 123")

```
TESTING 123, testing 123, TESTING 123, "testing 123", (testing  
123), testing 123.  
^  
Returns 1
```

ucInStr(1, MainString, "testing 123", **ucCase**)

```
TESTING 123, testing 123, TESTING 123, "testing 123", (testing  
123), testing 123.  
^  
Returns 16
```

ucInStr(1, MainString, "testing 123", **ucChar**)

```
TESTING 123, testing 123, TESTING 123, "testing 123", (testing  
123), testing 123.  
^  
Returns 31
```

ucInStr(1, MainString, "testing 123", **ucCase+ucChar**)

```
TESTING 123, testing 123, TESTING 123, "testing 123", (testing  
123), testing 123.  
^
```

Returns 47

ucInStr(1, MainString, "(a test)", ucCase+ucSpace)

TESTING 123, testing 123, TESTING 123, "testing 123", (testing 123), testing 123.

^
Returns 60

ucInStr(1, MainString, "(a test)", ucCase+ucSpace+ucBlock)

TESTING 123, testing 123, TESTING 123, "testing 123", (testing 123), testing 123.

^

Returns 74

Note: Not all combinations will produce a logically meaningful result. For instance ucChar cannot be combined with ucToken or ucBlock, because a character search is exclusive of token patterns.

ucLeft (*MainString*, *n* [, *Options*])

Returns the left-most *n* characters, tokens, blocks, or expressions in *MainString*.

Example: Left counting tokens (which is the default, and not characters)

ucLeft(MainString, 5)

This "is a test" using (the quick) brown fox, and more.

Returns: [*the text that's highlighted*]

Explanation: By default functions from this library count tokens (not characters). The 5 left-most tokens are:

1. This
2. "is a test"
3. using
4. (
5. the

This example is not spacing-sensitive.

Example: Left counting characters

ucLeft(MainString, 5, ucChar)

This "is a test" using (the quick) brown fox, and more.

Returns: "This " (without quotes, but including space)

Explanation: The 5 left-most characters are:

1. T
2. h
3. i
4. s
5. [space]

Example: Left counting blocks

ucLeft(MainString, 5, ucBlock)

This "is a test" using (the quick) brown fox, and more.

Returns: [*the text that's highlighted*]

Explanation: The 5 left-most blocks are:

1. This
2. "is a test"
3. using
4. (the quick)
5. brown

Example: Left counting tokens, space-sensitive

ucLeft(MainString, 5, ucSpace)

This "is a test" using (the quick) brown fox, and more.

Returns: [*the text that's highlighted*]

Explanation: The 5 left-most space-sensitive tokens are:

1. This
2. [space]
3. "is a test"
4. [space]
5. using

ucRight (*MainString*, *n* [, *Options*])

Returns the right-most *n* characters, tokens, blocks, or expressions in *MainString*.

The rules for `ucRight` are very similar to those for `ucLeft`. It is important to consider (especially if speed is a consideration) that unlike `ucLeft` which parses only until it reaches the number of characters or tokens specified, `ucRight` always parses the entire *MainString*, before returning the right-most *n* characters/tokens.

Examples: See ucLeft for the concept of counting by tokens, and blocks, etc.

ucLen (*MainString* [, *Options*])

Returns the length of a string in terms of characters, tokens, blocks, or expressions.

Example: counting tokens

ucLen(MainString)

This "is a test" using (the quick) brown fox, and more.

Returns: 13

Explanation: The counted tokens are:

1. This
2. "is a test"
3. Using
4. (
5. The
6. Quick
7.)
8. Brown
9. Fox
10. ,
11. And
12. More
13. .

Example: Len counting characters

ucLen(MainString, ucChar)

This "is a test" using (the quick) brown fox, and more.

Returns: 56

Explanation: This count each character much the same way VB's Len() function would.

Example: Len counting blocks

ucLen(MainString, ucBlock)

This "is a test" using (the quick) brown fox, and more.

Returns: [*the text that's highlighted*]

Explanation: The 5 left-most blocks are:

6. This
7. "is a test"
8. using
9. (the quick)
10. brown

Example: Len counting tokens (space-sensitive)

uLen(MainString, ucSpace)

This "is a test" using (the quick) brown fox, and more.

Returns: 21

Explanation: The counting here is similar to the example with ucChar except that each block of whitespace also counts as one unit. The space before "using" for instance counts as one, and the two consecutive spaces after "using" also count as one.

Example: Len counting using blocks

uLen(MainString, ucBlock)

This "is a test" using (the quick) brown fox, and more.

Returns: 10

Explanation: The text blocks that are counted are:

1. This
2. "is a test"
3. using
4. (the quick)
5. brown
6. fox
7. ,
8. and
9. more
10. .

ucUCase(MainString [, pattern] [, Options])

Returns an uppercase version of MainString.

Pattern – ucUCase without the optional arguments behaves the same way as VB's UCase() function. If the optional pattern argument is used, then only the matching pattern within MainString is changed to uppercase.

Example:

ucUCase(MainString, "{text}")

This "is a test" using (the quick) brown "fox", (and) more.

Returns:

This "is a test" using **(THE QUICK)** brown "fox", **(AND)** more.

ucUCase(MainString, "{Q}{text}{Q}")

This "is a test" using (the quick) brown "fox", (and) more.

Returns:

This **"IS A TEST"** using (the quick) brown **"FOX"**, (and) more.

ucUCase(MainString, "", ucSkipOver("{item}"))

This "is a test" using (the quick) brown "fox", (and) more.

Returns:

THIS "IS A TEST" USING (the quick) **BROWN "FOX"**, (and) **MORE.**

Explanation: Only text matching the pattern within the given string was changed to uppercase.

ucLCase(MainString [, pattern] [, Options])

Returns a lower case version of MainString.

See ucUCase.

ucMCase(MainString [, pattern] [, Options])

Returns a lower case version of MainString.

See ucUCase.

ucMid (MainString, Start [, Count] [, Options])

Returns a subset of a string.

Start – Character, token, block, or expression at which to start

Count – Length of substring to return in terms of characters, tokens, blocks, or expressionst.

Note: In this version, if the second arg is not optional; use -1 to make it go to the end. [to be fixed.]

Example:

ucMid(MainString, 6)

```
THIS IS A TEST, "This is a test", (a test), a test, a test.
^   ^   ^   ^   ^   ^
1   2   3   4   5   6
```

Returns

"This is a test", (a test), a test, a test.

ucMid(MainString, 6, 4)

```
THIS IS A TEST, "This is a test", (a test), a test, a test.
^   ^   ^   ^   ^   ^
1   2   3   4   5   6
```

Returns

"This is a test", (a

The 4 tokens counted are

1. "This is a test"
2. ,
3. (
4. a

ucMid(MainString, 6, 4, ucChar)

```
THIS IS A TEST, "This is a test", (a test), a test, a test.
^^^^^^
123456
```

ucReplace (*MainString, Pattern, Replacement [, Options]*)

Replaces all occurrences of one string pattern within MainString with another string.

Pattern – Pattern of text to find within MainString

Replacement – String or string pattern to replace text matching the pattern with.

Example: InStr with default setting

ucReplace(MainString, "is {tokens:2}", "was {tokens}?")

```
THIS IS A TEST, This is a test, (is a test).
```

Returns

THIS **was A TEST?** This **was a test?**, (**was a test?**).

ucReplace(MainString, "is {tokens:2}", "was {tokens}?", ucSkipOver("This {etc},"))

THIS IS A TEST, This is a test, (is a test).

Returns

THIS IS A TEST, This is a test, (**was a test?**).

Explanation: The first example skips over the part in quotes

ucTally(MainString, Pattern [, Options])

Counts the number of occurrences of text matching a given pattern within a string. The count can be based on characters, tokens, blocks, or expressions. You can select whether or not to make the result whitespace-sensitive.

Example:

ucTally(MainString, "is")

THIS **is** A TEST, "This is a test", (**is** a test).

Returns: 2

ucTally(MainString, "is", ucChar)

THIS **is is** A TEST, "Th**is is** a test", (**is** a test).

Returns: 5

ucTally(MainString, "is", ucSkipOver("{Q}"))

THIS **is** A TEST, "This **is** a test", (**is** a test).

Returns: 5

ucExtract(MainString, Pattern [, Options])

Extracts characters, tokens, blocks, or expressions up to a given pattern in a string. Similar to PowerBASIC's Extract, except that one extracts only characters.

Example:

ucExtract(MainString, "<{tag}>{word:2}</{tag}>")

This is <i>Test
</i>. This is another test. Testing 123.

Returns: This is <i>Test
</i>. This is

Explanation: returns text leading up to the first occurrence of text matching a pattern consisting of a two HTML-type tags, containing two words. It skips over Test which has only one word, and continues up until the second tag pair.

ucRemain(MainString, Pattern [, Options])

Returns text following a given pattern in a string. Similar to PowerBASIC's Extract, except that one only deals with characters.

Example:

ucRemain(MainString, "<{tag}>{anything}</{tag}>")

This is <i>Test
</i>. This is another test. Testing 123.

Returns: . This is another test. Testing 123.

Explanation: This returns text following (but not including) text that matches the pattern.

ucRetain(MainString, Pattern [, Options])

Returns a string consisting of only text that match a given pattern. Similar to PowerBASIC's Retain() function except that one deals only with characters.

Example:

ucRetain(ucFile("\Test\ucFile\cdcatalog.xml"), "<artist>{etc}</artist>", ucNth(22))

Returns: <artist>Luciano Pavarotti</artist>

Explanation: This example is meant to work with cdcatalog.xml, downloaded from http://www.w3schools.com/xml/cd_catalog.xml. ucFile() is a quick way to obtain all the text from one file. ucNth(22) tells it to retain only the 22th occurrence of the pattern of text between <artist> and </artist>.

Functions, which can be used in the Options argument:

ucStart(Position)
ucLimit(Limit)
ucStartAfter(number)
ucStopAfter(number)
ucBetween(a, b)
ucNth(number)
ucPos(x)
ucLength(x)
ucText(x)

Also

SetOptions(Options) – This sets defaults.