ucEval Function

See also ucEvaluate

Evaluates an expression in one step.

Syntax

ucEval(*expression*)

ucEval is a convenient one-step method for evaluating a math expression. It is adequate for general situations where the expression is not recalculated many times in a loop. It is the same as calling **ucParse**, and **ucEvaluate**, then returning a value and releasing the expression from the definition space. For loops which repeat the same expression, use ucParse and ucEvaluate separately for the fastest results. **ucEval** returns a value in double precision.

Example 1:

When you run this example, enter an expression such as: 5+4*10

```
UserExpression$ = InputBox("Enter a math expression")
MsgBox ucEval(UserExpression$)
```

Example 2:

Generally, your code should check for errors when **ucEval** is invoked. To test this example, create a form with a command button, and paste in the following code:

```
Sub Command1_Click()
    Dim UserExpression As String, Answer As Double

    On Error GoTo ErrorHandler

    UserExpression = InputBox("Enter a math expression")
    Answer = ucEval(UserExpression)

    If ucError Then
        MsgBox ucErrorMessage, vbExclamation
    Else
        MsgBox Answer, , "The answer is"
    End If
Exit Sub
ErrorHandler:
    MsgBox Error, vbExclamation
End Sub
```

ucEvaluate Function

See also ucEval, ucParse, and ucVariableValue

Rapidly evaluates an expression which has already been parsed.

Syntax

ucEvaluate(*ExpressionPtr***)**

In situations where the same expression is being evaluated repeatedly with changing values inside a loop, it is better to use **ucEvaluate** (instead of the one-step *ucEval* function) for much faster results. An expression must first be parsed with ucParse, which changes the string expression into simplified internal instructions for fast results and returns a pointer to the instructions representing the parsed expression. The returned pointer is a long integer which is to be used as the *ExpressionPtr* argument for **ucEvaluate**. This function returns a number in double precision.

Example:

This example uses the **ucEvaluate** function to do a simple summation of a user expression. Notice that the time-consuming **ucParse** function is used *before* the repetitious loop, and the fast **ucEvaluate** function is *inside* the loop. To run this example, place a command button on a form, and paste in the following code:

```
Private Sub Command1_Click()
    Dim x As Double, Total As Double, UserExpr As String
    Dim ExpressionPtr As Long, VariablePtr As Integer

    UserExpr = InputBox("Enter a math expression (such as x^2+5*x+14)")

    VariablePtr = ucDefineVariable("x")
    ExpressionPtr = ucParse(UserExpr)

    For x = 1 To 1000
        ucVariableValue(VariablePtr) = x
        Total = Total + ucEvaluate(ExpressionPtr)
    Next

    MsgBox "The total is: " & Total
    ucReleaseExpr
End Sub
```

ucParse Function

See also ucEvaluate, and ucReleaseExpr

Parses an expression and returns a pointer for use with ucEvaluate.

Syntax

ucParse(*expression***)**

The *expression* argument is a text string containing a math expression. **ucParse** changes the expression into simple internal instructions and returns a long integer which is a pointer to these instructions for subsequent use with ucEvaluate. This technique requires more programming steps than using the straightforward **ucEval** function, but it has a very substantial speed advantage when an expression must be re-evaluated many times in a loop. Parsing an expression is a relatively slow process because it handles text, and has many things to sort out. The result however is a simplified, and thus much faster set of instructions for **ucEvaluate** to handle. The slow process of parsing is done once, *before* starting the loop, and **ucEvaluate** which is *inside* the loop is left with a relatively light task for it to do.

Example:

This example uses the **ucEvaluate** function to do a simple summation of a user expression. Notice that the time-consuming **ucParse** function is used *before* the repetitious loop, and **ucEvaluate** is *inside* the loop. To run this example, place a command button on a form, and paste in the following code:

```
Private Sub Command1_Click()
    Dim x As Double, Total As Double, UserExpr As String
    Dim ExpressionPtr As Long, VariablePtr As Integer

    UserExpr = InputBox("Enter a math expression (such as x^2+5*x+14)")

    VariablePtr = ucDefineVariable("x")
    ExpressionPtr = ucParse(UserExpr)

    For x = 1 To 1000
        ucVariableValue(VariablePtr) = x
        Total = Total + ucEvaluate(ExpressionPtr)
    Next

    MsgBox "The total is: " & Total
    ucReleaseExpr
End Sub
```

ucDefineFunction Procedure

See also ucDefineVariable, and Technical Limitations

Defines, or redefines a user function.

Syntax

ucDefineFunction *userdef*

The *userdef* argument must consist of a text string which includes a function name, function parameter(s) enclosed in parenthesis (or empty parenthesis for no parameters), then an equal sign, followed by the function's definition. The naming convention is similar to that of VB. Function definitions can include previously defined user functions and variables.

Example:

```
ucDefineFunction "area(length,width) = length*width"
ucDefineFunction "frac(x)=abs(abs(x)-int(abs(x)))"
ucDefineFunction "test() =   z + 5"     ' assuming z is a variable already defined

MsgBox ucEval("frac(150/17) * area(20,30)")
```

ucDefineVariable Function

See also ucVariableValue, ucDefineFunction, and Technical Limitations

Defines, or assigns a value to a variable.

Syntax

ucDefineVariable(*userdef***)**

The *userdef* argument is a string which can consist of either: 1) a variable name by itself to be initialized, or 2) a variable name, then an equal sign followed by an expression which evaluates to the value to be assigned to the variable. The variable naming convention is similar to that of VB. Variable definitions may include previously defined user functions and variables.

ucDefineVariable returns an integer pointer for the variable. This pointer can be used as the argument for ucVariableValue, which also lets you assign a value to variable, but is faster because it is done more directly. In some cases you may not need this pointer. If that is the case, and if you are using VB, an alternative syntax is:

ucDefineVariable *userdef*

Note: 300 variables can be defined. Variables have a separate definition space, and unlike user functions, redefining the same variable does not take up additional space.

Example 1:

```
ucDefineVariable "pi = atn(1) * 4"
ucDefineVariable "MyVar = pi + 10"

MsgBox ucEval("3pi+MyVar / 2")
```

Example 2:

```
Dim VariableX As Integer, VariableY As Integer
Dim x As Double, y as Double, Total As Double, ExpressionPtr As Long

VariableX   = ucDefineVariable("x")
VariableY   = ucDefineVariable("y")

ExpressionPtr = ucParse("x^2+y^2")

For x = 0 To 100
    ucVariableValue(VariableX) = x
    For y = 0 To 100
        ucVariableValue(VariableY) = y
        Total = Total + ucEvaluate(ExpressionPtr)
    Next
Next

MsgBox Total
```

ucError Property

See also ucErrorMessage, and Error Handling

Returns an error number corresponding to the most recently parsed expression.

Syntax

ucError

The **ucError** property returns an integer value every time **ucParse**, **ucEval**, **ucDefineFunction**, or **ucDefineVariable** is called. A value of 0 means that the expression was properly parsed. Any other value indicates that the expression was not parsed due to an error in the expression. For the list of error codes, see ucErrorMessage.

Example:

```
MyExpr$ = "3 * (5 * 2"
ExprPtr = ucParse(MyExpr$)
If ucError Then MsgBox ucErrorMessage
' This will display: Mismatched parenthesis
```

Note: **ucError** does not handle errors for ucEvaluate (except for factorial overflow).

Error Handling

See also ucError, and ucErrorMessage

This component detects only parsing-related errors (with the exception of factorial overflow). Procedures which can return a parsing error are: ucEval, ucParse, ucDefineVariable, and ucDefineFunction. **ucError** returns a numerical value representing the error, and **ucErrorMessage** returns a text string containing the error message.

No error checking is done when expressions are evaluated (except for factorial overflow). Therefore, your program should include an error handling routine to trap *Division by Zero* and *Overflow* . Functions which require external error handling are: ucEval, ucEvaluate, and ucDefineVariable.

Introduction

See also Getting Started, License, and Contacting the Author

UCalc Fast Math Parser is an ActiveX component which allows programs to evaluate math expressions that are defined at run time. It includes a one step ucEval function for simplicity, as well as a very fast ucEvaluate function, designed for use in loops which repeat the same operation with different values. This component supports conventional math operators and functions, as well as user variables and functions. It is suitable for heavy duty number crunching, and uses the same robust code that is behind the highly rated UCALC for Windows program.

One Step Evaluation

An expression can be evaluated in just one step, using ucEval as in the following example:

```
UserExpression$ = InputBox("Enter a math expression")
' The user may enter something like:   5-(3+2)/8+10

Text1 = ucEval(UserExpression$)
```

Note: **ucEval** is designed for programming convenience, but should not be used in repetitions where speed is a concern. Instead, use **ucEvaluate** for greater speed.

Fast Evaluations

The following technique uses more steps than **ucEval**, but is designed for maximum speed. The ucParse function first parses an expression into simple instructions, and returns a pointer for the expression. Once parsed, subsequent evaluations using ucEvaluate will be very fast. Here's an example of how it works:

```
Dim x As Double, Total As Double, UserExpr As String
Dim ExpressionPtr As Long, VariablePtr As Integer

UserExpr = InputBox("Enter a math expression")
' The user may enter something like:   x^2+5*x+14

VariablePtr = ucDefineVariable("x")
ExpressionPtr = ucParse(UserExpr)

For x = 1 To 1000
    ucVariableValue(VariablePtr) = x
    Total = Total + ucEvaluate(ExpressionPtr)
Next

MsgBox "The total is: " & Total
ucReleaseExpr
```

User Functions and Variables

The following example shows how user-defined functions and variables are supported.

```
ucDefineFunction "area(length,width) = length*width"
ucDefineVariable "MyVar = 123"

Answer = ucEval("100*MyVar+area(5,20)")
```

Help file generated by VB HelpWriter.

ucErrorMessage Function

See also ucError, and Error Handling

Returns an error message.

Syntax

ucErrorMessage[(*errornumber*)]

This function returns a text string which describes a parsing error corresponding to the *errornumber* argument. This argument is an integer. If the optional argument is omitted, then a message associated with the most recent parsing error is returned. (The argument is not optional for VB 4.0 users. Use *errornumber* = -1 to get the most recent error).

Error Number	Error message
0	(None)
1	(Reserved)
2	Mismatched parenthesis
3	[Function/Variable] is not defined
4	Invalid binary number
5	Invalid octal number
6	Invalid hexadecimal number
7	Factorial overflow (returned by ucEvaluate)
8	(Reserved)
9	Invalid expression
10	Definition space is full
11	Invalid number of function parameters

Example:

```
MyExpr$ = "3 * (5 * 2"
ExprPtr = ucParse(MyExpr$)
If ucError Then MsgBox ucErrorMessage
' This will display: Mismatched parenthesis
```

Note: Errors such as *division by zero* and *overflow* should be handled by your program.

ucReset

See also ucReleaseExpr

Initializes or resets the math component.

Syntax

ucReset

When invoked, parsed expressions are released from the definition space, and user defined functions and variables are erased.

ucReleaseExpr Procedure

See also ucParse, and Technical Limitations

Releases expressions from the definition space.

Syntax:

ucReleaseExpr[(*number*)]

Every time ucParse is invoked, instructions for the parsed expression accumulate inside a certain definition space. An accumulation of too many parsed expressions will eventually cause the parser to run out of definition space. Once full, it will generally cause **ucError** to return error number 10 when ucParse, ucEval, ucDefineFunction, or ucDefineVariable are invoked. To avoid this, each **ucParse** statement should be released if no longer needed. **ucReleaseExpr** without the optional argument releases the most recently parsed expression. (The argument is not optional for VB 4.0 users). When the *number* argument is used, it represents the number of expressions to release, starting from the most recently defined one back to the oldest. The *number* argument is an integer.

A maximum of 200 expressions can be defined without having to be released. However, the definition space may fill up even before then, depending on the length -- in terms of number of instructions -- of all the parsed expressions combined. The maximum number of instructions is set at 1000. In a user expression being parsed, each operator, pair of parenthesis, or built-in function counts as one instruction, and a call to a user function takes up one instruction per argument, plus one for the function itself. **ucDefineFunction** also adds instructions to the same definition space.

If ucError returns an error number (non-zero), it means that the expression was not parsed, and no definition space was taken. **ucReleaseExpr** checks **ucError** first to make sure the recent expression was successfully parsed before attempting to release it. (Therefore it is not necessary to check for errors before invoking it).

Caution: Each **ucReleaseExpr** call should have matching expressions for it to release. It should not be used all by itself. If invoked without a recently parsed expression, it may corrupt the definition of recently defined user functions.

Example 1:

You will find this same example also under the help topic for ucEvaluate. Although it is not easy to actually measure it, the concept is that you can click on the command button as many times as you like without the definition space filling up, thanks to the **ucReleaseExpr** statement.

```
Private Sub Command1_Click()
    Dim x As Double, Total As Double, UserExpr As String
    Dim ExpressionPtr As Long, VariablePtr As Integer

    UserExpr = InputBox("Enter a math expression (such as x^2+5*x+14)")

    VariablePtr = ucDefineVariable("x")
    ExpressionPtr = ucParse(UserExpr)

    For x = 1 To 1000
        ucVariableValue(VariablePtr) = x
        Total = Total + ucEvaluate(ExpressionPtr)
    Next

    MsgBox "The total is: " & Total
```

```
    ucReleaseExpr
End Sub
```

Example 2:

ucReleaseExpr simply sets back the definition space pointer. This doesn't have an effect until the next expression is parsed (or a user function is defined), at which time the released instructions will be overwritten. Therefore the location of **ucReleaseExpr** in your code is not so important, as long as there is a matching expression for it to release. The following is a fragment from the previous example, with the difference being that **ucRelease** was placed right next to **ucParse** instead of at the end, to insure that the proper expression is released, as you add more lines of code to your project.

```
    VariablePtr = ucDefineVariable("x")
    ExpressionPtr = ucParse(UserExpr)
    ucReleaseExpr

    For x = 1 To 1000
        ucVariableValue(VariablePtr) = x
        Total = Total + ucEvaluate(ExpressionPtr)
    Next
```

Example 3:

This example demonstrates how the optional argument can be used in order to release several expressions at a time. It plots a parametric equation which requires two expressions to be parsed. In order to run this example, place a command button, two text boxes, and a picture box on a form. Enlarge the picture box to a comfortable viewing size. Then paste the following code in the General Declarations area. (This simple example doesn't remove the starting stray line).

```
Private Sub Form_Load()
    ' You can change these at runtime
    Text1 = "cos(t)-cos(2t)"
    Text2 = "sin(t)-sin(2t)"
End Sub

Private Sub Command1_Click()
    Dim UserEq1 As Long, UserEq2 As Long
    Dim theta As Double, thetaPtr As Integer
    Dim x As Double, y As Double

    Picture1.Cls
    Picture1.Scale (-2, 2)-(2, -2)

    thetaPtr = ucDefineVariable("t")
    UserEq1 = ucParse(Text1)
    UserEq2 = ucParse(Text2)
    ucReleaseExpr 2     ' Releases both UserEq1 & UserEq2 when done

    For theta = 0 To 6.28 Step 0.05
        ucVariableValue(thetaPtr) = theta
        x = ucEvaluate(UserEq1)
        y = ucEvaluate(UserEq2)

        Picture1.Line -(x, y)
    Next
End Sub
```

Help file generated by VB HelpWriter.

ucTrigMode Property

See also Operators

Sets or returns the trigonometric angle mode.

Syntax:

ucTrigMode [=*mode*]

Use this property to find out the current trigonometric mode, or to set it to one of the following:

1 Radian
2 Degree
3 Gradient

ucTrigMode affects the following functions: SIN, COS, TAN, SEC, CSC, COT, ASIN, ACOS, and ATAN. This property returns an integer. The default value is 1 for *radian* mode.

Example:

```
' Default mode is 1
MsgBox "TrigMode: " & ucTrigMode & "   sin(30)= " & ucEval("sin(30)")
ucTrigMode = 2          ' Sets the mode to degrees.
MsgBox "TrigMode: " & ucTrigMode & "   sin(30)= " & ucEval("sin(30)")
```

ucVariableValue Property

See also ucDefineVariable

Gets or sets a variable value directly.

Syntax

ucVariableValue(*VariablePtr***)** [= *value*]

The *VariablePtr* argument is an integer pointer for a variable which has been previously defined with ucDefineVariable. The assigned *value* is double precision. Although **ucDefineVariable** is also capable of assigning a value to a variable, it does so indirectly after parsing a text expression. **ucVariableValue** allows you to assign a value directly to the variable, making it the ideal companion for ucEvaluate when maximum speed is desired inside a loop.

Example:

This example plots a rough 3D graph, demonstrating the use of **ucVariableValue** inside a loop. In order to run this example, place a command button, a text box, and a picture box on a form. Enlarge the picture box to a comfortable viewing size. Then paste the following code in the General Declarations area.

```
Private Sub Form_Load()
    ' You can modify this during runtime
    Text1 = "(sin(x)+cos(z))*z/3"
End Sub

Private Sub Command1_Click()
    Dim x As Double, z As Double, UserEq As Long
    Dim VariableX As Integer, VariableZ As Integer

    Picture1.Cls
    Picture1.Scale (-10, 10)-(10, -10)

    VariableX = ucDefineVariable("x")
    VariableZ = ucDefineVariable("z")

    UserEq = ucParse(Text1)

    For z = -10 To 10 Step 0.5
        For x = -10 To 10 Step 0.5
            ucVariableValue(VariableX) = x
            ucVariableValue(VariableZ) = z
            Picture1.Line -((z - x) * 0.75, (ucEvaluate(UserEq) + z) * 0.75)
        Next
    Next
    ucReleaseExpr
End Sub
```

Built-in Functions and Operators

See also Precedence, and Notation

Symbol	Equivalent	Description	Example
( )		Prioritizes an expression	5*(1+1) = 10
!	FACT	Factorial	5! = 120
%		Percentage	35% = .35
^	**	Raised to the power of	4 ^ 5 = 1024
*		Multiply by	3 * 16 = 18
/		Divide by	9 / 2 = 4.5
MOD		Modulo (remainder)	7 MOD 4 = 3
+		Add	1 + 1 = 2
-		Subtract	9 - 5 = 4
>		Greater than	9 > 2 = 1 *see note
<		Less than	7 < 4 = 0
==	=	Equal test	5 == 4 = 0
>=	=>	Greater or equal	3 >= 3 = 1
<=	=<	Less or equal	#h3E <= 9 = 0
<>		Not equal	#b10101 <> 20 = 1
NOT		Bitwise NOT	NOT(15) = -16
AND	&	Bitwise AND	#b101 AND #h1E = 4
OR	|	Bitwise OR	13 OR 6 = 15
XOR		Bitwise Exclusive OR	9 XOR 3 = 10
EQV		Bitwise Equivalence	6 EQV 9 = -16
IMP		Bitwise Implication	1 IMP 5 = -1
SIN		Sine	sin(pi) = 0 *see note
COS		Cosine	cos(pi) = -1
TAN		Tangent	tan(pi) = 0
ASIN		Arcsine	asin(1) = 1.570
ACOS		Arcosine	acos(-1) = 3.141
ATAN	ATN	Arctangent	atan(0) = 0
SINH		Hyperbolic sine	sinh(3) = 10.01
COSH		Hyperbolic cosine	cosh(2) = 3.76
TANH		Hyperbolic tangent	tanh(1) = 0.76
COTH		Hyperbolic cotangent	coth(1) = 1.31
SECH		Hyperbolic secant	sech(0) = 1
CSCH		Hyperbolic cosecant	csch(1) = 0.85
ASINH		Hyperbolic arcsine	asinh(2) = 1.44
ACOSH		Hyperbolic arccosine	acosh(9) = 2.89
ATANH		Hyperbolic arctangent	atanh(.1) = 0.10
ACOTH		Hyperbolic arccotangent	acoth(7) = 0.14
ASECH		Hyperbolic arcsecant	asech(.3) = 1.87

ACSCH		Hyperbolic arccosecant	acsch(2) = 0.48
ABS		Absolute value	abs(-8) = 8
EXP		e to the power of	exp(3) = 20.08
EXP2		2 to the xth power	exp2(3) = 8
EXP10		10 to the xth power	exp10(3) = 1000
CEIL		Round up	ceil(6.2) = 7
RND		Random number	rnd(1) = .969
INT		Truncate to an integer	int(6.8) = 6
FACT	!	Factorial	fact(5) = 120
LOG	LN	Natural log	log(16) = 2.77
LOG2		Log base 2	log2(8) = 3
LOG10		Log base 10	log10(100) = 2
SGN	SIGN	Sign of expression (-1, 0 or 1)	sgn(-9) = -1
SQR	SQRT	Square root	sqr(64) = 8

Note: Relational operators, (>, <, <=, >=, ==, <>) return a 1 or a 0 (for true or false)

The trig examples in the above table assume that "pi" was defined.

(CSC, SEC, and COT are supported as well)

How to Contact the Author

See also License

The author will be happy to hear comments, or answer questions about licensing for the UCalc Fast Math Parser component. You may contact Daniel Corbier using one of the following methods:

E-mail
FastMath@ucalc.com (Internet)
Dancorbier (AOL)
75541,1523 (Compuserve)

Note: E-mail is by far the preferred mode of communication with the author (especially if it requires a quick response). When sending e-mail, please include a descriptive subject, so that your message does not get confused with the large amount of spam coming in. Include key words such as: ucalc, math, or parser, etc... to make sure it doesn't slip my attention.

Phone/Fax: 305-233-2604

Postal
Daniel Corbier
20410 SW 92 Place
Miami FL 33189, USA

Web Page
Be sure to visit **http://www.ucalc.com/dll** from time to time to check up on new information and updates.

Operator Order of Precedence

See also Operators, and Notation

Here is the precedence list from highest to lowest priority:

Anything inside parenthesis is performed first	( )
Factorial, percentage	!, %
Exponentiation	^
Multiplication, division	*, /
Modulo (remainder)	MOD
Addition, subtraction	+, -
Relational operators	<, >, >=, <=, =, <>
AND operator	
OR, XOR (exclusive or)	
EQV (equivalence)	
IMP (implication)	

When consecutive operators have the same priority, UCALC evaluates from left to right. This means that an expression such as "a-b-c" is evaluated as "(a-b)-c".

UCALC for Windows

See also Contacting the Author

The technology used in this component is the same one behind the highly rated UCALC for Windows user program. UCALC contains various tools for performing calculations, all combined into one powerful yet simple program. Here is a highlight of some of its features:

- **Expression Evaluator** which supports a number of built-in functions, operators, numeric bases, numeric formats, and modes.
- **Unit Converter**. Allows you to convert between units of measure in a flexible manner. It also includes a world currency unit category.
- **User Solution Modules**. This innovative feature allows users to interactively define and solve problems by simply filling in the blanks without directly having to manipulate math formulas.
- **User Functions & Variables**
- **Equation Solver** which allows you to solve for the unknown value in an algebraic expression.
- **Summation Table** is a general mathematical tool which can be used for practical things such as calculating annuity, compound growth, etc...
- **Numerical Integration**.
- **Graphing**. UCALC can plot Cartesian, polar, parametric, 3D equations, or coordinates from data files.
- **General Ledger**. UCALC is not just for heavy duty scientific calculations. This simple general ledger provides a convenient way for adding up many numbers.

To download an evaluation copy of this program, or to find out more about it, visit **http://www.ucalc.com** .

License Agreement

See also Contacting the Author

This document serves as a license agreement between Daniel Corbier and you (as an individual or single entity software developer using this component), pertaining to this TRIAL version of the UCalc Fast Math Parser component. For information on how to obtain a different license, visit **http://www.ucalc.com/dll** . Your usage of this component implies your acceptance to abide by the terms of this end user license agreement.

1. GRANT OF LICENSE. Daniel Corbier grants to you a non-exclusive, non-transferable license to install and use the software on your computer. This license is conditional, based on the following limitations: (i) You may not reverse engineer, decompile, disassemble, or otherwise attempt to modify the component (ii) You may not remove, suppress, hide, or cause your program to skip the opening message boxes (iii) **You may not use this TRIAL copy for purposes other than evaluation of the component, or beta testing of your own product**. Visit **http://www.ucalc.com/dll** for information on obtaining a different license.

2. DISTRIBUTION. This component may be distributed under these conditions: (i) All parts of the component (including documentation and sample project files) must be distributed together. (ii) The component and associated files may be distributed as a unit on conventional shareware distribution sites (iii) The component may be distributed as part of another program which uses it, ONLY if it is part of a beta test version for which no money is charged, requested, or accepted. To find out how to obtain a different license visit **http://www.ucalc.com/dll** .

3. NOT FOR RESALE: You may not resell, or otherwise transfer for value this component. You may not rent, lease, or lend this component. You may not include this trial copy in a version of a software product that is being sold or as part of software which helps generate revenue either directly or indirectly. Shareware/freeware distributors however, may include the component as part of a larger collection for which a nominal fee is charged, as long as the fee is for the collection and not the component itself.

4. TECHNICAL SUPPORT. The component author may provide support on a voluntary basis, but will not be bound to providing support for the trial copy. The author will not be bound to provide an immediate program fix for this trial copy, even in the event that a problem has been brought to his attention.

5. LIMITED WARRANTY. This component is provided on an "as is" basis without warranty of any kind, expressed or implied, including, but not limited to implied warranties of merchantability or fitness for a particular purpose. All other warranties are also expressly disclaimed.

6. SUITABILITY. The persons or companies using this component are solely responsible for thoroughly testing it to make sure it is works to their satisfaction before incorporating a non-evaluation version into their programs. All features are made available for testing purposes in this evaluation copy.

7. EXPORT. You agree to comply with U.S. export laws and regulations concerning the acquisition, usage or transfer of computer software.

8. GOVERNMENT RESTRICTIONS. This software and documentation are subject to Restricted Rights, as expressed under applicable regulations.

Notation

See also Operators, and Precedence

The syntax for expressions accepted by this component is similar to that of VB. This component also supports alternative notations, as shown in the list of built-in operators, as well as the following enhancements:

Implicit Multiplication

When multiplication is implied, the times symbol (*) can often be omitted, as in the following examples:

Expression	**Equivalent**
x y	x*y
3pi + 10	3*pi + 10
5(4+8)	5*(4+8)
(5+5)(3+9)	(5+5)*(3+9)
(3+2)8	(3+2)*8

Note: Implicit multiplication has the same priority as regular multiplication. For instance '1/2q' is translated as '1/2*q' not '1/(2q)'.

Numeric Bases

This component supports notations for binary, octal, and hexadecimal numbers. These numbers must be preceded by the character # followed by b, o, or h for binary, octal, or hexadecimal.

Example:

UserExpression = "#b110101110 AND #h1AE OR #o656"

Technical Limitations

See also ucReleaseExpr, and ucReset

The following technical limits should be taken into account while using this component:

- A maximum of 300 user variables can be defined. Once defined a variable cannot be released (except with ucReset).

- A maximum of 200 user functions can be defined. Unlike variables which are each stored as one value, each function uses up some of the definition space in proportion to the length of the function definition (in terms of number of instructions). It is possible to run out of definition space before defining 200 functions. Functions cannot be released (except with ucReset). Unlike with variables, when the same function is redefined it takes up additional definition space.

- A maximum of 200 different parsed expressions can be concurrently active. Parsed expressions take up definition space in the same manner as user functions. Unlike functions however, parsed expressions can be released from the definition space, using ucReleaseExpr. It is a good idea to make use of it, to avoid eventually filling up the definition space.

- The definition space stores a maximum of 1000 instructions. Every time an expression is parsed without being released, or a user function is defined or redefined, it accumulates instructions in the definition space. In a user expression, each operator, pair of parenthesis, or built-in function counts as one instruction, and each call to a user function takes up one instruction per argument, plus one for the function itself. Once the definition space is completely full, attempts to parse will return error number 10 in ucError. ucReleaseExpr and ucReset can be used to release instructions from the definition space.

- Maximum speed generally is not obtained while running the program in the VB IDE (although it will still be relatively fast). The parser's routines run faster from a compiled program.

Getting Started

See also Introduction, and Contacting the Author

This document is written primarily with Visual Basic 5.0 users in mind. The component itself however, should be compatible with any compiler which supports 32-bit ActiveX technology. If you are using another compiler (including VB4) you should make sure you have up-to-date copies of VB 5.0 runtime files which are listed at the bottom.

Installation

To install this component automatically, run the SETUP.EXE program. This will place the files in the proper Windows directory and registry. You may also install the component manually in a temporary directory by decompressing each file individually with EXPAND.EXE (or an equivalent utility). Then register the .DLL files with REGSVR32.EXE (or an equivalent utility).

Once the component is installed, in order to use it in your project, VB 5.0 users should go to the Project menu and select References. Click on the box next to *UCALC Fast Math Parser* . VB 4.0 users will find the component under the Tools / References menu selection. After that, you are ready to add Fast Math code to your program. Other compilers may require different steps.

Context Sensitive Help and Examples

UCalc Fast Math Parser comes with a sample executable (ucSample.EXE) which demonstrate most of the procedures, along with the source code for it (vb4sampl.* for VB4 and vb5sampl.* for VB5). A good way to familiarize yourself with the procedures is to click on the Object Browser icon. Select *UcalcFastMath*, and you will see the list of all the procedures with a short description for each one along with the required arguments and data types. Select an item on the list, and click on the Help icon (which has a question mark) for more information. Pressing F1 on a component keyword from within your source code will also provide help for the desired item. Compilers other than VB may have a different way of doing this.

Component Files

Files which are part of this component:

README.TXT	Information about this component
FILE_ID.DIZ	Brief description for shareware distributors
LICENSE.TXT	License agreement and disclaimer
UCALC.DEP	Component dependency file
UCALC.DLL	ActiveX component
UCALCDLL.HLP	Help file
UCALCDLL.CNT	Help file content list
VB4SAMPL.FRM	Visual Basic 4.0 sample form
VB4SAMPL.VBP	Visual Basic 4.0 sample project file
VB5SAMPL.FRM	Visual Basic 5.0 sample form
VB5SAMPL.VBP	Visual Basic 5.0 sample project file
UCSAMPLE.EXE	Executable demo

Runtime files required by this component:

MSVBVM50.DLL
STDOLE2.TLB
OLEAUT32.DLL
OLEPRO32.DLL
ASYCFILT.DLL

CTL3D32.DLL
COMCAT.DLL

These runtime files may have come with your copy of this component's setup. If not please visit **http://www.ucalc.com/dll** to obtain them. If you obtain them from elsewhere, make sure the files are up-to-date (check UCALC.DEP for file dates).