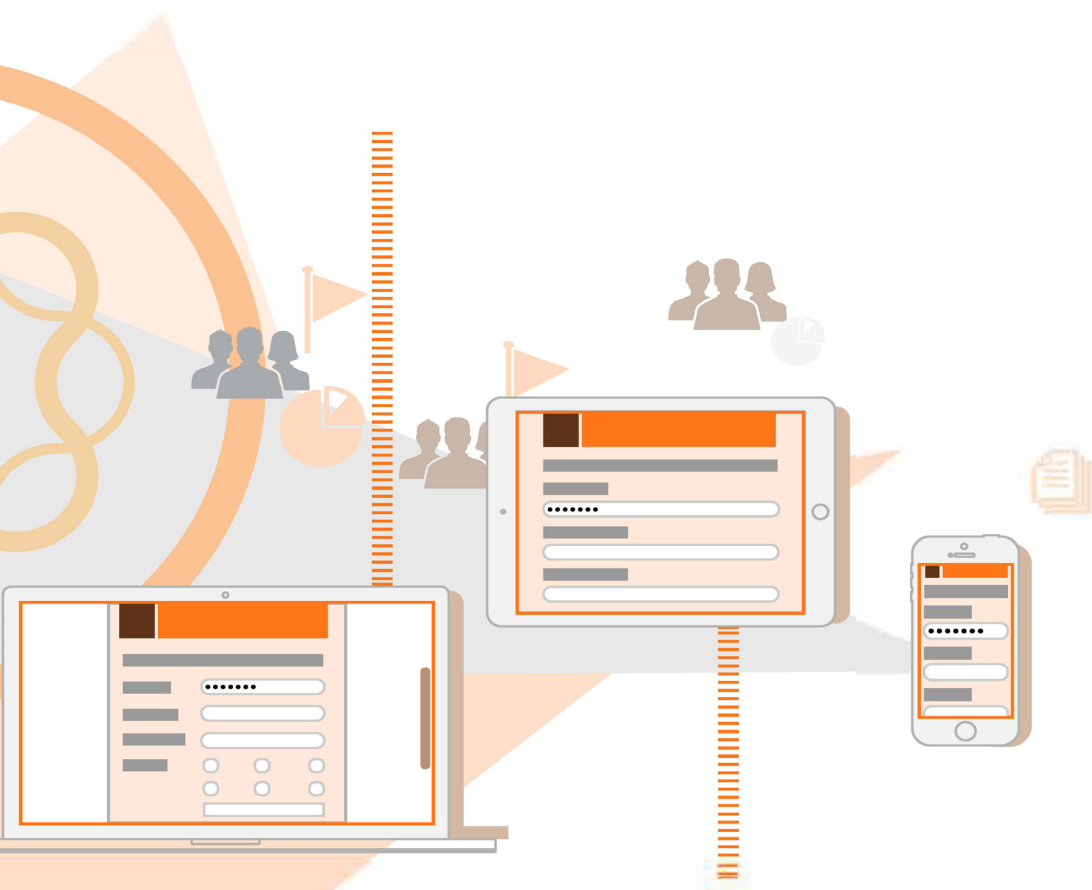

Designer FormCalc Reference



AEM 6.5 Forms

Legal notices

For legal notices, see http://help.adobe.com/en_US/legalnotices/index.html.

Contents

About FormCalc	1
Using FormCalc in Designer	1
Expressions	2
Simple	2
About simple expressions	2
Promoting operands	3
Numeric operations	3
Boolean operations	4
String operations	4
Assignment	5
Logical OR	5
Logical AND	5
Unary	6
Equality and inequality	7
Relational	8
If expressions	9
While expressions	10
For expressions	11
Foreach expressions	12
Break expressions	12
Continue expressions	13
Building blocks	14
Literals	14
Operators	16
Comments	17
Keywords	18
Identifiers	18

Line terminators	19
White space	19
Variables	20
Reference Syntax	20
Current field or object	21
Data model root of xfa.datasets.data	21
Form object event	21
Form model root	22
Host object	22
Layout model root	22
Collection of data record	23
Template model root	23
Data model root of xfa.datasets	24
Select all form objects	24
Search for objects that are part of a subcontainer	24
Denote an unnamed object or specify a property	25
Occurrence value of an object	26
Property and method calls	27
Built-in function calls	28
 Alphabetical Functions List	 29
 Arithmetic Functions	 33
Abs	33
Syntax	33
Parameters	33
Examples	33
Avg	34
Syntax	34
Parameters	34
Examples	34
Ceil	35
Syntax	35
Parameters	35
Examples	35
Count	35
Syntax	36
Parameters	36
Examples	36
Floor	36
Syntax	36
Parameters	37
Examples	37
Max	37
Syntax	37

Parameters	37
Examples	38
Min	38
Syntax	38
Parameters	39
Examples	39
Mod	39
Syntax	40
Parameters	40
Examples	40
Round	41
Syntax	41
Parameters	41
Examples	41
Sum	42
Syntax	42
Parameters	42
Examples	42
Date and Time Functions	43
Structuring dates and times	43
Epoch	43
Date formats	43
Time formats	44
Date and time picture formats	45
Locales	49
Date	49
Syntax	49
Parameters	49
Examples	49
Date2Num	50
Syntax	50
Parameters	50
Examples	50
DateFmt	51
Syntax	51
Parameters	51
Examples	51
IsoDate2Num	52
Syntax	52
Parameters	52
Examples	52
IsoTime2Num	53
Syntax	53
Parameters	53
Examples	53

LocalDateFmt	54
Syntax	54
Parameters	54
Examples	54
LocalTimeFmt	55
Syntax	55
Parameters	55
Examples	55
Num2Date	56
Syntax	56
Parameters	56
Examples	56
Num2GMTTime	57
Syntax	57
Parameters	57
Examples	57
Num2Time	58
Syntax	58
Parameters	58
Examples	59
Time	59
Syntax	59
Parameters	59
Examples	59
Time2Num	60
Syntax	60
Parameters	60
Examples	60
TimeFmt	61
Syntax	61
Parameters	61
Examples	62
Financial Functions	63
Apr	63
Syntax	63
Parameters	63
Examples	63
CTerm	64
Syntax	64
Parameters	64
Examples	65
FV	65
Syntax	65
Parameters	65
Examples	66

IPmt66
Syntax67
Parameters67
Examples67
NPV68
Syntax68
Parameters68
Examples68
Pmt69
Syntax69
Parameters69
Examples70
PPmt70
Syntax70
Parameters70
Examples71
PV71
Syntax72
Parameters72
Examples72
Rate73
Syntax73
Parameters73
Examples73
Term74
Syntax74
Parameters74
Examples75
Logical Functions76
Choose76
Syntax76
Parameters76
Examples76
Exists77
Syntax77
Parameters77
Examples77
HasValue78
Syntax78
Parameters78
Examples78
Oneof79
Syntax79
Parameters79
Examples79

Within	80
Syntax	80
Parameters	80
Examples	80
Miscellaneous Functions	82
Eval	82
Syntax	82
Parameters	82
Examples	82
Null	83
Definition	83
Parameters	83
Examples	83
Ref	83
Definition	83
Parameters	84
Examples	84
UnitType	84
Syntax	84
Parameters	85
Examples	85
UnitValue	86
Syntax	86
Parameters	86
Examples	86
String Functions	88
At	88
Syntax	88
Parameters	88
Examples	88
Concat	89
Syntax	89
Parameters	89
Examples	89
Decode	89
Syntax	90
Parameters	90
Examples	90
Encode	90
Syntax	91
Parameters	91
Examples	91
Format	91

Syntax	91
Parameters	92
Examples	93
Left	93
Syntax	93
Parameters	93
Examples	93
Len	94
Syntax	94
Parameters	94
Examples	94
Lower	95
Syntax	95
Parameters	95
Examples	95
Ltrim	96
Syntax	96
Parameters	96
Examples	96
Parse	97
Syntax	97
Parameters	97
Examples	97
Replace	98
Syntax	98
Parameters	98
Examples	98
Right	99
Syntax	99
Parameters	99
Examples	99
Rtrim	100
Syntax	100
Parameters	100
Examples	100
Space	100
Syntax	100
Parameters	101
Examples	101
Str	101
Syntax	101
Parameters	101
Examples	102
Stuff	102
Syntax	102
Parameters	103

Examples	103
Substr	103
Syntax	104
Parameters	104
Examples	104
Uuid	105
Syntax	105
Parameters	105
Examples	105
Upper	105
Syntax	106
Parameters	106
Examples	106
WordNum	106
Syntax	107
Parameters	107
Examples	107
URL Functions	109
Get	109
Syntax	109
Parameters	109
Examples	109
Post	110
Syntax	110
Parameters	110
Examples	111
Put	112
Syntax	112
Parameters	112
Examples	113

About FormCalc

Designer provides users with FormCalc, a simple yet powerful calculation language for those with little or no scripting experience.

Designer provides users with FormCalc, a simple yet powerful calculation language for those with little or no scripting experience.

The form developer can incorporate calculations and scripting to create a richer experience for the recipient of the form. FormCalc facilitates fast and efficient form design without requiring a knowledge of traditional scripting techniques or languages.

The built-in functions in FormCalc cover a wide range of areas including mathematics, dates and times, strings, finance, logic, and the web. These areas represent the types of data that typically occur in forms, and the functions provide quick and easy manipulation of the data in a useful way.

NOTE: This document uses terms Adobe Experience Manager Forms, AEM Forms, AEM Forms on JEE, and LiveCycle interchangeably.

Using FormCalc in Designer

Within Designer, JavaScript™ is the default scripting language, with FormCalc as the alternative. Scripting takes place on the various events that accompany each form object, and you can use a mixture of JavaScript and FormCalc in interactive forms.

If you are using a server-based process, such as forms generator, to create forms for viewing in an internet browser, FormCalc scripts on certain form object events do not render onto the HTML form. This functionality is intended to prevent Internet browser errors from occurring when users work with the completed form.

RELATED LINKS:

[Alphabetical Functions List](#)

Expressions

Literals, operators, comments, keywords, identifiers, line terminators, and white space come together to form a list of expressions, even if the list only contains a single expression. In general, each expression in the list resolves to a value, and the value of the list as a whole is the value of the last expression in the list.

For example, consider the following scenario of two fields on a form design:

Field name	Calculations	Returns
Field1	5 + Abs(Price) "Hello World" 10 * 3 + 5 * 4	50
Field2	10 * 3 + 5 * 4	50

The value of both `Field1` and `Field2` after the evaluation of each field's expression list is 50.

FormCalc divides the various types of expressions that make up an expression list into the following categories:

Simple

About simple expressions

In their most basic form, FormCalc expressions are groups of operators, keywords, and literals strung together in logical ways. For example, these are all simple expressions:

```
2
"abc"
2 - 3 * 10 / 2 + 7
```

Each FormCalc expression resolves to a single value by following a traditional order of operations, even if that order is not always obvious from the expression syntax. For example, the following sets of expressions, when applied to objects on a form design, produce equivalent results:

Expression	Equivalent to	Returns
"abc"	"abc"	abc
2 - 3 * 10 / 2 + 7	2 - (3 * (10 / 2)) + 7	-6

Expression	Equivalent to	Returns
10 * 3 + 5 * 4	(10 * 3) + (5 * 4)	50
0 and 1 or 2 > 1	(0 and 1) or (2 >1)	1(true)
2 < 3 not 1 == 1	(2 < 3) not (1 == 1)	0(false)

As the previous table suggests, all FormCalc operators carry a certain precedence when they appear within expressions. The following table illustrates this operator hierarchy:

Precedence	Operator
Highest	=
	(Unary) -, +, not
	*, /
	+, -
	<, <=, >, >=, lt, le, gt, ge
	==, <>, eq, ne
	&, and
Lowest	, or

Promoting operands

In cases where one or more of the operands within a given operation do not match the expected type for that operation, FormCalc promotes the operands to match the required type. How this promotion occurs depends on the type of operand required by the operation.

Numeric operations

When performing numeric operations involving non-numeric operands, the non-numeric operands are first promoted to their numeric equivalent. If the non-numeric operand does not successfully convert to a numeric value, its value is 0. When promoting null-valued operands to numbers, their value is always zero.

The following table provides some examples of promoting non-numeric operands:

Expression	Equivalent to	Returns
(5 - "abc") * 3	(5 - 0) * 3	15
"100" / 10e1	100 / 10e1	1
5 + null + 3	5 + 0 + 3	8

Boolean operations

When performing Boolean operations on non-Boolean operands, the non-Boolean operands are first promoted to their Boolean equivalent. If the non-Boolean operand does not successfully convert to a nonzero value, its value is true (1); otherwise its value is false (0). When promoting null-valued operands to a Boolean value, that value is always false (0). For example, the expression:

```
"abc" | 2
```

evaluates to 1. That is, false | true = true, whereas

```
if ("abc") then
10
else
20
endif
```

evaluates to 20.

String operations

When performing string operations on nonstring operands, the nonstring operands are first promoted to strings by using their value as a string. When promoting null-valued operands to strings, their value is always the empty string. For example, the expression:

```
concat("The total is ", 2, " dollars and ", 57, " cents.")
```

evaluates to "The total is 2 dollars and 57 cents."

NOTE: If during the evaluation of an expression an intermediate step yields NaN, +Inf, or -Inf, FormCalc generates an error exception and propagates that error for the remainder of the expression. As such, the expression's value will always be 0. For example:

```
3 / 0 + 1
```

evaluates to 0.

Assignment

An assignment expression sets the property identified by a given reference syntax to be the value of a simple expression. For example:

```
$template.purchase_order.name.first = "Tony"
```

This sets the value of the form design object “first” to Tony.

For more information on using syntax, see [Referencing Objects in Calculations and Scripts](#)

For more information on using syntax, see [Referencing Objects in Calculations and Scripts](#)

Logical OR

A logical OR expression returns either true (1) if at least one of its operands is true (1), or false (0) if both operands are false (0). If both operands are null, the expression returns null.

Expression	Character representation
Logical OR	 or

These are examples of using the logical OR expression:

Expression	Returns
1 or 0	1(true)
0 0	0(false)
0 or 1 0 or 0	1(true)

Logical AND

A logical AND expression returns either true (1) if both operands are true (1), or false if at least one of its operands is false (0). If both operands are null, the expression returns null.

Expression	Character representation
Logical AND	& and

These are examples of using the logical AND expression:

Expression	Returns
1 and 0	0(false)
0 & 0	1(true)
0 and 1 & 0 and 0	0(false)

Unary

A unary expression returns different results depending on which of the unary operators is used.

Expression	Character representation	Returns
Unary	-	The arithmetic negation of the operand, or null if the operand is null.
	+	The arithmetic value of the operand (unchanged), or null if its operand is null.
	not	The logical negation of the operand.

NOTE: The arithmetic negation of a null operand yields the result null, whereas the logical negation of a null operand yields the Boolean result true. This is justified by the common sense statement: If null means nothing, then “not nothing” should be something.

These are examples of using the unary expression:

Expression	Returns
- (17)	-17

Expression	Returns
<code>- (-17)</code>	17
<code>+ (17)</code>	17
<code>+ (-17)</code>	-17
<code>not ("true")</code>	1 (true)
<code>not (1)</code>	0 (false)

Equality and inequality

Equality and inequality expressions return the result of an equality comparison of its operands.

Expression	Character representation	Returns
Equality	<code>==</code> <code>eq</code>	True (1) when both operands compare identically, and false (0) if they do not compare identically.
Inequality	<code><></code> <code>ne</code>	True (1) when both operands do not compare identically, and false (0) if they compare identically.

The following special cases also apply when using equality operators:

- If either operand is null, a null comparison is performed. Null-valued operands compare identically whenever both operands are null, and compare differently whenever one operand is not null.
- If both operands are references, both operands compare identically when they both refer to the same object, and compare differently when they do not refer to the same object.
- If both operands are string valued, a locale-sensitive lexicographic string comparison is performed on the operands. Otherwise, if they are not both null, the operands are promoted to numeric values, and a numeric comparison is performed.

These are examples of using the equality and inequality expressions:

Expression	Returns
<code>3 == 3</code>	1 (true)

Expression	Returns
3 <> 4	1 (true)
"abc" eq "def"	0 (false)
"def" ne "abc"	1 (true)
5 + 5 == 10	1 (true)
5 + 5 <> "10"	0 (false)

Relational

A relational expression returns the Boolean result of a relational comparison of its operands.

Expression	Character representation	Returns
Relational	< lt	True (1) when the first operand is less than the second operand, and false (0) when the first operand is larger than the second operand.
	> gt	True (1) when the first operand is greater than the second operand, and false (0) when the first operand is less than the second operand.
	<= le	True (1) when the first operand is less than or equal to the second operand, and false (0) when the first operand is greater than the second operand.
	>= ge	True (1) when the first operand is greater than or equal to the second operand, and false (0) when the first operand is less than the second operand.

The following special cases also apply when using relational operators:

- If either operand is null valued, a null comparison is performed. Null-valued operands compare identically whenever both operands are null and the relational operator is less-than-or-equal or greater than or equal, and compare differently otherwise.
- If both operands are string valued, a locale-sensitive lexicographic string comparison is performed on the operands. Otherwise, if they are not both null, the operands are promoted to numeric values, and a numeric comparison is performed.

These are examples of using the relational expression:

Expression	Returns
3 < 3	0(false)
3 > 4	0(false)
"abc" <= "def"	1(true)
"def" > "abc"	1(true)
12 >= 12	1(true)
"true" < "false"	0(false)

If expressions

An if expression is a conditional statement that evaluates a given simple expression for truth, and then returns the result of a list of expressions that correspond to the truth value. If the initial simple expression evaluates to false (0), FormCalc examines any elseif and else conditions for truth and returns the results of their expression lists if appropriate.

Expression	Syntax	Returns
If	<pre>if (simple expression) then list of expressions elseif (simple expression) then list of expressions else list of expressions endif</pre>	<p>The result of the list of expressions associated with any valid conditions stated in the if expression.</p> <p>You are not required to have any elseif(...) or else statements as part of your if expression, but you must state the end of the expression with endif.</p>

These are examples of using the if expression:

Expression	Returns
<pre>if (1 < 2) then 1 endif</pre>	1

Expression	Returns
<pre>if ("abc" > "def") then 1 and 0 else 0 endif</pre>	0
<pre>if (Field1 < Field2) then Field3 = 0 elseif (Field1 > Field2) then Field3 = 40 elseif (Field1 == Field2) then Field3 = 10 endif</pre>	Varies with the values of Field1 and Field2. For example, if Field1 is 20 and Field2 is 10, then this expression sets Field3 to 40.

While expressions

A while expression is an iterative statement or loop that evaluates a given simple expression. If the result of the evaluation is true (1), FormCalc repeatedly examines the do condition and returns the results of the expression lists. If the result is false (0), then control passes to the next statement.

A while expression is particularly well-suited to situations in which conditional repetition is needed. Conversely, situations in which unconditional repetition is needed are often best dealt with using a for expression.

Expression	Syntax	Returns
While	<pre>while (simple expression) do expression list endwhile</pre>	The result of the list of expressions associated with the do condition.

In the following example, the values of the elements are added to a drop-down list from an XML file using the addItem method for all of the XML elements listed under list1 that are not equal to 3:

```
var List = ref(xfa.record.lists.list1)
var i = 0
while ( List.nodes.item(i+1).value ne "3")do
$.addItem (List.nodes.item(i).value,List.nodes.item(i+1).value)
i = i + 2
endwhile
```

For expressions

A for expression is a conditionally iterative statement or loop.

A for expression is particularly well-suited to looping situations in which unconditional repetition is needed. Conversely, situations in which conditional repetition is needed are often best dealt with using a while expression.

The value of the for expression is the value of the last evaluation list that was evaluated, or false (0).

The for condition initializes a FormCalc variable, which controls the looping action.

In the upto variant, the value of the loop variable will iterate from the start expression to the end expression in step expression increments. If you omit the step expression, the step increment defaults to 1.

In the downto variant, the value of the loop variable iterates from the start expression to the end expression in step expression decrements. If the step expression is omitted, the step decrements defaults to -1.

Iterations of the loop are controlled by the end expression value. Before each iteration, the end expression is evaluated and compared to the loop variable. If the result is true (1), the expression list is evaluated. After each evaluation, the step expression is evaluated and added to the loop variable.

Before each iteration, the end expression is evaluated and compared to the loop variable. In addition, after each evaluation of the do condition, the step expression is evaluated and added to the loop variable.

A for loop terminates when the start expression has surpassed the end expression. The start expression can surpass the end expression in either an upwards direction, if you use upto, or in a downward direction, if you use downto.

Expression	Syntax	Returns
For	<pre>for variable = start expression (upto downto) end expression (step step expression) do expression list endfor</pre> <p>The start, end, and step expressions must all be simple expressions.</p>	The result of the list of expressions associated with the do condition.

In the following example, the values of the elements are added to a drop-down list from an XML file using the addItem method for all of the XML elements listed under list1:

```
var List = ref(xfa.record.lists.list1)
for i=0 upto List.nodes.length - 1 step 2 do
```

```
$.addItem (List.nodes.item(i).value, "")
endfor
```

Foreach expressions

A foreach expression iterates over the expression list for each value in its argument list.

The value of the foreach expression is the value of the last expression list that was evaluated, or zero (0), if the loop was never entered.

The in condition, which is executed only once (after the loop variable has been declared) controls the iteration of the loop. Before each iteration, the loop variable is assigned successive values from the argument list. The argument list cannot be empty.

Expres sion	Syntax	Returns
Foreac h	<pre>foreach variable in(argument list)do expression list endfor</pre> Use a comma (,) to separate more than one simple expression in the argument list.	The value of the last expression list that was evaluated, or zero(0), if the loop was never entered.

In the following example, only the values of the “display” XML elements are added to the foreach drop-down list.

```
foreach Item in (xfa.record.lists.list1.display[*]) do
$.addItem(Item, "")
endfor
```

Break expressions

A break expression causes an immediate exit from the innermost enclosing while, for, or foreach expression loop. Control passes to the expression following the terminated loop.

The value of the break expression is always the value zero (0).

Expression	Syntax	Returns
Break	<code>break</code>	Passes control to the expression following the terminated loop.

In the following example, an if condition is placed in the while loop to check whether the current value is equal to “Display data for 2”. If true, the break executes and stops the loop from continuing.

```
var List = ref(xfa.record.lists.list1)
var i=0
while (List.nodes.item(i+1).value ne "3") do
$.addItem(List.nodes.item(i).value,List.nodes.item(i+1).value)
i = i + 2
if (List.nodes.item(i) eq "Display data for 2" then
break
endif
endwhile
```

Continue expressions

A continue expression causes the next iteration of the innermost enclosing while, for, or foreach loop.

The value of the continue expression is always the value zero (0).

Expression	Syntax	Returns
Continue	<code>continue</code>	When used in a while expression, control is passed to the while condition. When used in a for expression, control is passed to the step expression.

The object of the following example is to populate the drop-down list with values from the XML file. If the value of the current XML element is “Display data for 3,” then the while loop exits via the break expression. If the value of the current XML element is “Display data for 2”, then the script adds 2 to the variable `i` (which is the counter) and immediately the loop moves on to its next cycle. The last two lines are ignored when the value of the current XML element is “Display data for 2”.

```
var List = ref(xfa.record.lists.list1)
var i = 0
while (List.nodes.item(i+1).value ne "5") do
if (List.nodes.item(i) eq "Display data for 3") then
break
endif
if (List.nodes.item(i) eq "Display data for 2" then
```

```
i=i+2
continue
endif
$.addItem(List.nodes.item(i).value,List.nodes.item(i+1).value)
i=i+2
endwhile
```

Building blocks

The FormCalc language consists of building blocks that make up FormCalc expressions. Each FormCalc expression is a sequence of some combination of these building blocks.

Literals

Literals are constant values that form the basis of all values that pass to FormCalc for processing. The two general types of literals are numbers and strings.

Number literals

A number literal is a sequence of mostly digits consisting of one or more of the following characters: an integer, a decimal point, a fractional segment, an exponent indicator (“e” or “E”), and an optionally signed exponent value. These are all examples of literal numbers:

- -12
- 1.5362
- 0.875
- 5.56e-2
- 1.234E10

It is possible to omit either the integer or fractional segment of a literal number, but not both. In addition, within the fractional segment, you can omit either the decimal point or the exponent value, but not both.

All number literals are internally converted to Institute of Electrical and Electronics Engineers (IEEE) 64-bit binary values. However, IEEE values can only represent a finite quantity of numbers, so certain values do not have a representation as a binary fraction. This is similar to the fact that certain values, such as $1/3$, do not have a precise representation as a decimal fraction (the decimal value would need an infinite number of decimal places to be entirely accurate).

The values that do not have a binary fraction equivalent are generally number literals with more than 16 significant digits prior to their exponent. FormCalc rounds these values to the nearest representable IEEE 64-bit value in accordance with the IEEE standard. For example, the value:

```
123456789.012345678
```

rounds to the (nearest) value:

```
123456789.01234567
```

However, in a second example, the number literal:

```
999999999999999999
```

rounds to the (nearest) value:

```
1000000000000000000
```

This behavior can sometimes lead to surprising results. FormCalc provides a function, Round, which returns a given number rounded to a given number of decimal places. When the given number is exactly halfway between two representable numbers, it is rounded away from zero. That is, the number is rounded up if positive and down if negative. In the following example:

```
Round(0.124, 2)
```

returns 0.12,

and

```
Round(.125, 2)
```

returns 0.13.

Given this convention, one might expect that:

```
Round(0.045, 2)
```

returns 0.05.

However, the IEEE 754 standard dictates that the number literal 0.045 be approximated to 0.04499999999999999. This approximation is closer to 0.04 than to 0.05. Therefore,

```
Round(0.045, 2)
```

returns 0.04.

This also conforms to the IEEE 754 standard.

IEEE 64-bit values support representations like NaN (not a number), +Inf (positive infinity), and -Inf (negative infinity). FormCalc does not support these, and expressions that evaluate to NaN, +Inf, or -Inf result in an error exception, which passes to the remainder of the expression.

String literals

A string literal is a sequence of any Unicode characters within a set of quotation marks. For example:

```
"The cat jumped over the fence."
```

```
"Number 15, Main street, California, U.S.A"
```

The string literal "" defines an empty sequence of text characters called the empty string.

To embed a quotation mark (") character within a literal string, you must insert two quotation marks. For example:

```
"The message reads: ""Warning: Insufficient Memory"""
```

All Unicode characters have an equivalent 6 character escape sequence consisting of \u followed by four hexadecimal digits. Within any literal string, it is possible to express any character, including control characters, using their equivalent Unicode escape sequence. For example:

```
"\u0047\u0066\u0066\u0069\u0073\u0068\u0021"  
"\u000d" (carriage return)  
"\u000a" (newline character)
```

Operators

FormCalc includes a number of operators: unary, multiplicative, additive, relational, equality, logical, and the assignment operator.

Several of the FormCalc operators have an equivalent mnemonic operator keyword. These keyword operators are useful whenever FormCalc expressions are embedded in HTML and XML source text, where the symbols less than (<), greater than (>), and ampersand (&) have predefined meanings and must be escaped. The following table lists all FormCalc operators, illustrating both the symbolic and mnemonic forms where appropriate.

Operator type	Representations
Addition	+
Division	/
Equality	== eq <> ne
Logical AND	& and
Logical OR	or
Multiplication	*
Relational	< lt (less than) > gt (greater than) <= le (less than or equal to) >= ge (greater than or equal to)
Subtraction	-
Unary	- + not

Comments

Comments are sections of code that FormCalc does not execute. Typically comments contain information or instructions that explain the use of a particular fragment of code. FormCalc ignores all information stored in comments at run time.

You can specify a comment by using either a semi-colon (;) or a pair of slashes (//). In FormCalc, a comment extends from its beginning to the next line terminator.

Character name	Representations
Comment	; //

For example:

```
// This is a type of comment
First_Name="Tony"
Initial="C" ;This is another type of comment
Last_Name="Blue"
```

Commenting all FormCalc calculations on an event

Commenting all of the FormCalc calculations for a particular event generates an error when you preview your form in the Preview PDF tab or when you view the final PDF. Each FormCalc calculation is required to return a value, and FormCalc does not consider comments to be values.

To prevent the commented FormCalc code from returning an error, you must do one of the following actions:

- Remove the commented code from the event
- Add an expression that returns a value to the FormCalc code on the event

To prevent the value of the expression from producing unwanted results on your form, use one of the following types of expressions:

- A simple expression consisting of a single character, as shown in the following example:

```
// First_Name="Tony"
// Initial="C"
// Last_Name="Blue"
//
// The simple expression below sets the value of the event to zero.
0
```

- An assignment expression that retains the value of the object. Use this type of expression if your commented FormCalc code is located on the calculate event to prevent the actual value of the object from being altered, as shown in the following example:

```
// First_Name="Tony"
// Initial="C"
// Last_Name="Blue"
//
// The assignment expression below sets the value of the current
// field equal to itself.
$.rawValue = $.rawValue
```

Keywords

Keywords in FormCalc are reserved words and are case-insensitive. Keywords are used as parts of expressions, special number literals, and operators.

The following table lists the FormCalc keywords. Do not use any of these words when naming objects on your form design.

and	endif	in	step
break	endwhile	infinity	then
continue	eq	le	this
do	exit	lt	throw
downto	for	nan	upto
else	foreach	ne	var
elseif	func	not	while
end	ge	null	
endfor	gt	or	
endfunc	if	return	

Identifiers

An identifier is a sequence of characters of unlimited length that denotes either a function or a method name. An identifier always begins with one of the following characters:

- Any alphabetic character (based on the Unicode letter classifications)
- Underscore (_)
- Dollar sign (\$)
- Exclamation mark (!)

FormCalc identifiers are case-sensitive. That is, identifiers whose characters only differ in case are considered distinct.

Character name	Representations
Identifier	A..Z,a..z \$! —

These are examples of valid identifiers:

```
GetAddr  
$primary  
_item  
!dbresult
```

Line terminators

Line terminators are used for separating lines and improving readability.

The following table lists the valid FormCalc line terminators:

Character name	Unicode characters
Carriage Return	#xD U+000D
Line Feed	#xA  &#D;

White space

White space characters separate various objects and mathematical operations from each other. These characters are strictly for improving readability and are irrelevant during FormCalc processing.

Character name	Unicode character
Form Feed	#xC
Horizontal Tab	#x9

Character name	Unicode character
Space	#x20
Vertical Tab	#xB

Variables

Within your calculations, FormCalc allows you to create and manipulate variables for storing data. The name you assign to each variable you create must be a unique Identifier.

For example, the following FormCalc expressions define the `userName` variable and set the value of a text field to be the value of `userName`.

```
var userName = "Tony Blue"
TextField1.rawValue = userName
```

You can reference variables that you define in the Variables tab of the Form Properties dialog box in the same way. The following FormCalc expression uses the `Concat` function to set the value of the text field using the form variables `salutation` and `name`.

```
TextField1.rawValue = Concat(salutation, name)
```

NOTE: A variable you create using FormCalc will supersede a similarly named variable you define in the Variables tab of the Form Properties dialog box.

Reference Syntax

FormCalc provides access to form design object properties and values using a reference syntax. The following example demonstrates both assigning and retrieving object values:

```
Invoice_Total.rawValue = Invoice_SubTotal.rawValue * (8 / 100)
```

In this case the reference syntax `Invoice_Total` assigns the value of `Invoice_SubTotal * (8 / 100)` to the field `Invoice_Total`.

In the context of form design, a fully qualified reference syntax enables access to all the objects on a form design.

To make accessing object properties and values easier, FormCalc includes reference syntax shortcuts to reduce the effort required to create references.

Current field or object

Refers to the current field or object

Notation

`$`

Example

```
$ = "Tony Blue"
```

The above example sets the value of the current field or object to Tony Blue.

Data model root of `xfa.datasets.data`

Represents the root of the data model `xfa.datasets.data`.

Notation

`$data`

Example

```
$data.purchaseOrder.total
```

is equivalent to

```
xfa.datasets.data.purchaseOrder.total
```

Form object event

Represents the current form object event.

Notation

`$event`

Example

```
$event.name
```

is equivalent to

```
xfa.event.name
```

For more information, see [Working with the Event Model](#).

For more information, see [Working with the Event Model](#).

Form model root

Represents the root of the form model `xfa.form`.

Notation

```
$form
```

Example

```
$form.purchaseOrder.tax
```

is equivalent to stating

```
xfa.form.purchaseOrder.tax
```

Host object

Represents the host object.

Notation

```
$host
```

Example

```
$host.messageBox("Hello world")
```

is equivalent to

```
xfa.host.messageBox("Hello world")
```

For more information, see [Working with a Host Application](#)

For more information, see [Working with a Host Application](#)

Layout model root

Represents the root of the layout model `xfa.layout`.

Notation

`$layout`

Example

`$layout.ready`

is equivalent to stating

`xfa.layout.ready`

Collection of data record

Represents the current record of a collection of data, such as from an XML file.

Notation

`$record`

Example

`$record.header.txtOrderedByCity`

references the `txtOrderedByCity` node within the header node of the current XML data.

Template model root

Represents the root of the template model `xfa.template`.

Notation

`$template`

Example

`$template.purchaseOrder.item`

is equivalent to

`xfa.template.purchaseOrder.item`

Data model root of `xfa.datasets`

Represents the root of the data model `xfa.datasets`.

Notation

!

Example

`!data`

is equivalent to

`xfa.datasets.data`

Select all form objects

Selects all form objects within a given container, such as a subform, regardless of name, or selects all objects that have a similar name.

You can use the ‘*’ (asterisk) syntax with JavaScript if it used with the `resolveNode` method.

Notation

*

Example

For example, the following expression selects all objects named `item` on a form:

`xfa.form.form1.item[*]`

Search for objects that are part of a subcontainer

You can use two dots at any point in your reference syntax to search for objects that are a part of any subcontainer of the current container object, such as a subform.

You can use the ‘..’ (double period) syntax with JavaScript if it used with the `resolveNode` method.

Notation

..

Example

The expression `Subform_Page..Subform2` means locate the node `Subform_Page` (as usual) and find a descendant of `Subform_Page` called `Subform2`.



Using the example tree above,

`Subform_Page..TextField2`

is equivalent to

`Subform_Page.Subform1[0].Subform3.TextField2[0]`

because `TextField2[0]` is in the first `Subform1` node that `FormCalc` encounters on its search. As a second example,

`Subform_Page..Subform3[*]`

returns all four `TextField2` objects.

Denote an unnamed object or specify a property

The number sign (#) notation is used to denote one of the following items in a reference syntax:

- An unnamed object
- Specify a property in a reference syntax if a property and an object have the same name

You can use the '#' (number sign) syntax with JavaScript if it used with the `resolveNode` method.

Notation

#

Example

The following reference syntax accesses an unnamed subform:

`xfa.form.form1.#subform`

The following reference syntax accesses the `name` property of a subform if the subform also contains a field named `name`:

`xfa.form.form1.#subform.#name`

Occurrence value of an object

The square bracket ([]) notation denotes the occurrence value of an object.

In language-specific forms for Arabic, Hebrew, Thai, and Vietnamese, the reference syntax is always on the right (even for right-to-left languages).

Notation

[]

Example

To construct an occurrence value reference, place square brackets ([]) after an object name, and enclose within the brackets one of the following values:

- [n], where *n* is an absolute occurrence index number beginning at 0. An occurrence number that is out of range does not return a value. For example,

```
xfa.form.form1.#subform.Quantity[3]
```

refers to the fourth occurrence of the Quantity object.

- [+/- n], where *n* indicates an occurrence relative to the occurrence of the object making the reference. Positive values yield higher occurrence numbers, and negative values yield lower occurrence numbers. For example,

```
xfa.form.form1.#subform.Quantity[+2]
```

This reference yields the occurrence of Quantity whose occurrence number is two more than the occurrence number of the container making the reference. For example, if this reference was attached to the Quantity[2] object, the reference would be the same as

```
xfa.template.Quantity[4]
```

If the computed index number is out of range, the reference returns an error.

The most common use of this syntax is for locating the previous or next occurrence of a particular object. For example, every occurrence of the Quantity object (except the first) might use Quantity[-1] to get the value of the previous Quantity object.

- [*] indicates multiple occurrences of an object. The first named object is found, and objects of the same name that are siblings to the first are returned. Note that using this notation returns a collection of objects. For example,

```
xfa.form.form1.#subform.Quantity[*]
```

- This expression refers to all objects with a name of `Quantity` that are siblings to the first occurrence of `Quantity` found by the reference.



Using the tree for reference, these expressions return the following objects:

- `Subform_Page.Subform1[*]` returns both `Subform1` objects.
- `Subform_Page.Subform1.Subform3.TextField2[*]` returns two `TextField2` objects. `Subform_Page.Subform1` resolves to the first `Subform1` object on the left, and `TextField2[*]` evaluates relative to the `Subform3` object.
- `Subform_Page.Subform1[*].Textfield1` returns both of the `Textfield1` instances. `Subform_Page.Subform1[*]` resolves to both `Subform1` objects, and `Textfield1` evaluates relative to the `Subform1` objects.
- `Subform_Page.Subform1[*].Subform3.TextField2[1]` returns the second and fourth `TextField2` objects from the left. `Subform_Page.Subform1[*]` resolves to both `Subform1` objects, and `TextField2[1]` evaluates relative to the `Subform3` objects.
- `Subform_Page.Subform1[*].Subform3[*]` returns both instances of the `Subform3` object.
- `Subform_Page.*` returns both `Subform1` objects and the `Subform2` object.
- `Subform_Page.Subform2.*` returns the two instances of the `NumericField2` object.
- You can use the ' `[]`' (square bracket) syntax with JavaScript if it used with the `resolveNode` method.

Property and method calls

Designer defines a variety of properties and methods for all objects on a form design. FormCalc provides access to these properties and methods and allows you to use them to modify the appearance and behavior of objects on your form. Similar to a function call, you invoke properties and methods by passing arguments to them in a specific order. The number and type of arguments in each property and method are specific to each object type.

NOTE: Different form design objects support different properties and methods. For a complete list of the properties and methods objects support, see [About the Scripting Reference](#).

NOTE: Different form design objects support different properties and methods. For a complete list of the properties and methods objects support, see [About the Scripting Reference](#).

Built-in function calls

FormCalc supports a large set of built-in functions with a wide range of capabilities. The names of the functions are case-insensitive, but unlike keywords, FormCalc does not reserve the names of the functions. This means that calculations on forms with objects whose names coincide with the names of FormCalc functions do not conflict.

Functions may or may not require some set of arguments to execute and return a value. Many functions have arguments that are optional, meaning it is up to you to decide if the argument is necessary for the particular situation.

FormCalc evaluates all function arguments in order, beginning with the lead argument. If an attempt is made to pass less than the required number of arguments to a function, the function generates an error exception.

Each function expects each argument in a particular format, either as a number literal or string literal. If the value of an argument does not match what a function expects, FormCalc converts the value. For example:

```
Len (35)
```

The Len function actually expects a literal string. In this case, FormCalc converts the argument from the number 35 to the string "35", and the function evaluates to 2.

However, in the case of a string literal to number literal, the conversion is not so simple. For example:

```
Abs ("abc")
```

The Abs function expects a number literal. FormCalc converts the value of all string literals as 0. This can cause problems in functions where a 0 value forces an error, such as in the case of the Apr function.

Some function arguments only require integral values; in such cases, the passed arguments are always promoted to integers by truncating the fractional part.

Here is a summary of the key properties of built-in functions:

- Built-in function names are case-insensitive.
- The built-in functions are predefined, but their names are not reserved words. This means that the built-in function Max never conflicts with an object, object property, or object method named Max.
- Many of the built-in functions have a mandatory number of arguments, which can be followed by an optional number of arguments.
- A few built-in functions, Avg, Count, Max, Min, Sum, and Concat, accept an indefinite number of arguments.

For a complete listing of all the FormCalc functions, see the [Alphabetical FunctionsList](#).

Alphabetical Functions List

The following table lists all available FormCalc functions, provides a description of each function, and identifies the category type to which each function belongs.

Function	Description	Type
Abs	Returns the absolute value of a numeric value or expression.	Arithmetic
Apr	Returns the annual percentage rate for a loan.	Financial
At	Locates the starting character position of a string within another string.	String
Avg	Evaluates a set of number values and/or expressions and returns the average of the non-null elements contained within that set.	Arithmetic
Ceil	Returns the whole number greater than or equal to a given number.	Arithmetic
Choose	Selects a value from a given set of parameters.	Logical
Concat	Returns the concatenation of two or more strings.	String
Count	Evaluates a set of values and/or expressions and returns the number of non-null elements contained within the set.	Arithmetic
CTerm	Returns the number of periods needed for an investment earning a fixed, but compounded, interest rate to grow to a future value.	Financial
Date	Returns the current system date as the number of days since the Epoch	Date and Time
Date2Num	Returns the number of days since the Epoch, given a date string.	Date and Time
DateFmt	Returns a date format string, given a date format style.	Date and Time
Decode	Returns the decoded version of a given string.	String
Encode	Returns the encoded version of a given string.	String
Eval	Returns the value of a given form calculation.	Miscellaneous

Function	Description	Type
Exists	Determines whether the given parameter is a reference syntax to an existing object.	Logical
Floor	Returns the largest whole number that is less than or equal to the given value.	Arithmetic
Format	Formats the given data according to the specified picture format string.	String
FV	Returns the future value of consistent payment amounts made at regular intervals at a constant interest rate.	Financial
Get	Downloads the contents of the given URL.	URL
HasValue	Determines whether the given parameter is an accessor with a non-null, non-empty, or non-blank value.	Logical
IPmt	Returns the amount of interest paid on a loan over a set period of time.	Financial
IsoDate2Num	Returns the number of days since the Epoch, given an valid date string.	Date and Time
IsoTime2Num	Returns the number of milliseconds since the Epoch, given a valid time string.	Date and Time
Left	Extracts a specified number of characters from a string, starting with the first character on the left.	String
Len	Returns the number of characters in a given string.	String
LocalDateFmt	Returns a localized date format string, given a date format style.	Date and Time
LocalTimeFmt	Returns a localized time format string, given a time format style.	Date and Time
Lower	Converts all uppercase characters within a specified string to lowercase characters.	String
Ltrim	Returns a string with all leading white space characters removed.	String
Max	Returns the maximum value of the non-null elements in the given set of numbers.	Arithmetic
Min	Returns the minimum value of the non-null elements of the given set of numbers.	Arithmetic

Function	Description	Type
Mod	Returns the modulus of one number divided by another.	Arithmetic
NPV	Returns the net present value of an investment based on a discount rate and a series of periodic future cash flows.	Financial
Null	Returns the null value. The null value means no value.	Miscellaneous
Num2Date	Returns a date string, given a number of days since the Epoch.	Date and Time
Num2GMTTime	Returns a GMT time string, given a number of milliseconds from the Epoch.	Date and Time
Num2Time	Returns a time string, given a number of milliseconds from the Epoch.	Date and Time
Oneof	Returns true (1) if a value is in a given set, and false (0) if it is not.	Logical
Parse	Analyzes the given data according to the given picture format.	String
Pmt	Returns the payment for a loan based on constant payments and a constant interest rate.	Financial
Post	Posts the given data to the specified URL.	URL
PPmt	Returns the amount of principal paid on a loan over a period of time.	Financial
Put	Uploads the given data to the specified URL.	URL
PV	Returns the present value of an investment of periodic constant payments at a constant interest rate.	Financial
Rate	Returns the compound interest rate per period required for an investment to grow from present to future value in a given period.	Financial
Ref	Returns a reference to an existing object.	Miscellaneous
Replace	Replaces all occurrences of one string with another within a specified string.	String
Right	Extracts a number of characters from a given string, beginning with the last character on the right.	String

Function	Description	Type
Round	Evaluates a given numeric value or expression and returns a number rounded to the given number of decimal places.	Arithmetic
Rtrim	Returns a string with all trailing white space characters removed.	String
Space	Returns a string consisting of a given number of blank spaces.	String
Str	Converts a number to a character string. FormCalc formats the result to the specified width and rounds to the specified number of decimal places.	String
Stuff	Inserts a string into another string.	String
Substr	Extracts a portion of a given string.	String
Sum	Returns the sum of the non-null elements of a given set of numbers.	Arithmetic
Term	Returns the number of periods needed to reach a given future value from periodic constant payments into an interest-bearing account.	Financial
Time	Returns the current system time as the number of milliseconds since the Epoch.	Date and Time
Time2Num	Returns the number of milliseconds since the Epoch, given a time string.	Date and Time
TimeFmt	Returns a time format, given a time format style.	Date and Time
UnitType	Returns the units of a unitspan. A unitspan is a string consisting of a number followed by a unit name.	Miscellaneous
UnitValue	Returns the numeric value of a measurement with its associated unitspan, after an optional unit conversion.	Miscellaneous
Upper	Converts all lowercase characters within a string to uppercase.	String
Uuid	Returns a Universally Unique Identifier (UUID) string to use as an identification method.	String
Within	Returns true (1) if the test value is within a given range, and false (0) if it is not.	Logical
WordNum	Returns the English text equivalent of a given number.	String

Arithmetic Functions

Arithmetic functions perform a range of mathematical operations.

Abs

Returns the absolute value of a numeric value or expression, or returns null if the value or expression is null.

Syntax

`Abs (n1)`

Parameters

Parameter	Description
n1	A numeric value or expression to evaluate.

NOTE: *FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [Numberliterals](#).*

Examples

The following expressions are examples of using the `Abs` function:

Expression	Returns
<code>Abs (1.03)</code>	1.03
<code>Abs (-1.03)</code>	1.03
<code>Abs (0)</code>	0

Avg

Evaluates a set of number values and/or expressions and returns the average of the non-null elements contained within that set.

Syntax

```
Avg(n1 [, n2 ...])
```

Parameters

Parameter	Description
n1	The first numeric value or expression of the set.
n2(optional)	Additional numeric values or expressions.

NOTE: *FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [Numberliterals](#).*

Examples

The following expressions are examples of using the Avg function:

Expression	Returns
Avg(0, 32, 16)	16
Avg(2.5, 17, null)	9.75
Avg(Price[0], Price[1], Price[2], Price[3])	The average value of the first four non-null occurrences of Price.
Avg(Quantity[*])	The average value of all non-null occurrences of Quantity.

Ceil

Returns the whole number greater than or equal to a given number, or returns null if its parameter is null.

Syntax

`Ceil (n)`

Parameters

Parameter	Description
n	Any numeric value or expression. The function returns 0 if n is not a numeric value or expression.

***NOTE:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [Number literals](#).*

Examples

The following expressions are examples of using the `Ceil` function:

Expression	Returns
<code>Ceil (2.5875)</code>	3
<code>Ceil (-5.9)</code>	-5
<code>Ceil ("abc")</code>	0
<code>Ceil (A)</code>	100 if the value of A is 99.999

Count

Evaluates a set of values and/or expressions and returns the count of non-null elements contained within the given set.

Syntax

`Count(n1 [, n2 ...])`

Parameters

Parameter	Description
n1	A numeric value or expression.
n2(optional)	Additional numeric values and/or expressions.

NOTE: *FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [Numberliterals](#).*

Examples

The following expressions are examples of using the `Count` function:

Expression	Returns
<code>Count("Tony", "Blue", 41)</code>	3
<code>Count(Customers[*])</code>	The number of non-null occurrences of <code>Customers</code> .
<code>Count(Coverage[2], "Home", "Auto")</code>	3, provided the third occurrence of <code>Coverage</code> is non-null.

Floor

Returns the largest whole number that is less than or equal to the given value.

Syntax

`Floor(n)`

Parameters

Parameter	Description
n	Any numeric value or expression.

NOTE: *FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [Numberliterals](#).*

Examples

The following expressions are examples of using the `Floor` function:

Expression	Returns
<code>Floor(21.3409873)</code>	21
<code>Floor(5.999965342)</code>	5
<code>Floor(3.2 * 15)</code>	48

Max

Returns the maximum value of the non-null elements in the given set of numbers.

Syntax

```
Max(n1 [, n2 ...])
```

Parameters

Parameter	Description
n1	A numeric value or expression.
n2(optional)	Additional numeric values and/or expressions.

NOTE: *FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [Numberliterals](#).*

Examples

The following expressions are examples of using the `Max` function:

Expression	Returns
<code>Max(234, 15, 107)</code>	234
<code>Max("abc", 15, "Tony Blue")</code>	15
<code>Max("abc")</code>	0
<code>Max(Field1[*], Field2[0])</code>	Evaluates the non-null occurrences of <code>Field1</code> as well as the first occurrence of <code>Field2</code> , and returns the highest value.
<code>Max(Min(Field1[*], Field2[0]), Field3, Field4)</code>	The first expression evaluates the non-null occurrences of <code>Field1</code> as well as the first occurrence of <code>Field2</code> , and returns the lowest value. The final result is the maximum of the returned value compared against the values of <code>Field3</code> and <code>Field4</code> . See also <code>Min</code> .

Min

Returns the minimum value of the non-null elements of the given set of numbers.

Syntax

```
Min(n1 [, n2 ...])
```

Parameters

Parameter	Description
n1	A numeric value or expression.
n2(optional)	Additional numeric values and/or expressions.

NOTE: *FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [Numberliterals](#).*

Examples

The following expressions are examples of using the `Min` function:

Expression	Returns
<code>Min(234, 15, 107)</code>	15
<code>Min("abc", 15, "Tony Blue")</code>	15
<code>Min("abc")</code>	0
<code>Min(Field1[*], Field2[0])</code>	Evaluates the non-null occurrences of <code>Sales_July</code> as well as the first occurrence of <code>Sales_August</code> , and returns the lowest value.
<code>Min(Max(Field1[*], Field2[0]), Field3, Field4)</code>	The first expression evaluates the non-null occurrences of <code>Field1</code> as well as the first occurrence of <code>Field2</code> , and returns the highest value. The final result is the minimum of the returned value compared against the values of <code>Field3</code> and <code>Field4</code> . See also <code>Max</code> .

Mod

Returns the modulus of one number divided by another. The modulus is the remainder of the division of the dividend by the divisor. The sign of the remainder always equals the sign of the dividend.

Syntax

`Mod (n1, n2)`

Parameters

Parameter	Description
n1	The dividend, a numeric value or expression.
n2	The divisor, a numeric value or expression.

If n1 and/or n2 are not numeric values or expressions, the function returns 0.

NOTE: *FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [Number literals](#).*

Examples

The following expressions are examples of using the `Mod` function:

Expression	Returns
<code>Mod (64, -3)</code>	1
<code>Mod (-13, 3)</code>	-1
<code>Mod ("abc", 2)</code>	0
<code>Mod (X[0], Y[9])</code>	The first occurrence of <code>X</code> is used as the dividend and the tenth occurrence of <code>Y</code> is used as the divisor.
<code>Mod (Round (Value[4], 2), Max (Value[*]))</code>	The first fifth occurrence of <code>Value</code> rounded to two decimal places is used as the dividend and the highest of all non-null occurrences of <code>Value</code> is used as the divisor. See also Max and Round .

Round

Evaluates a given numeric value or expression and returns a number rounded to a given number of decimal places.

Syntax

`Round(n1 [, n2])`

Parameters

Parameter	Description
n1	A numeric value or expression to be evaluated.
n2(optional)	The number of decimal places with which to evaluate n1 to a maximum of 12. If you do not include a value for n2, or if n2 is invalid, the function assumes the number of decimal places is 0.

NOTE: FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see *Number literals*.

Examples

The following expressions are examples of using the Round function:

Expression	Returns
<code>Round(12.389764537, 4)</code>	12.3898
<code>Round(20/3, 2)</code>	6.67
<code>Round(8.9897, "abc")</code>	9
<code>Round(FV(400, 0.10/12, 30*12), 2)</code>	904195.17. This takes the value evaluated using the FV function and rounds it to two decimal places. See also FV.
<code>Round(Total_Price, 2)</code>	Rounds off the value of Total_Price to two decimal places.

Sum

Returns the sum of the non-null elements of a given set of numbers.

Syntax

```
Sum(n1 [, n2 ...])
```

Parameters

Parameter	Description
n1	A numeric value or expression.
n2(optional)	Additional numeric values and/or expressions.

NOTE: FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see *Numberliterals*.

Examples

The following expressions are examples of using the Sum function:

Expression	Returns
Sum(2, 4, 6, 8)	20
Sum(-2, 4, -6, 8)	4
Sum(4, 16, "abc", 19)	39
Sum(Amount[2], Amount[5])	Totals the third and sixth occurrences ofAmount.
Sum(Round(20/3, 2), Max(Amount[*]), Min(Amount[*]))	Totals the value of 20/3 rounded to two decimal places, as well as the largest and smallest non-null occurrences ofAmount. See also Max, Min, and Round.

Date and Time Functions

Date and time functions deal specifically with creating and manipulating date and time values.

Structuring dates and times

Epoch

Date values and time values have an associated origin or *epoch*, which is a moment in time from which time begins. Any date value and any time value prior to its epoch is invalid.

The unit of value for all date functions is the number of days since the epoch. The unit of value for all time functions is the number of milliseconds since the epoch.

Designer defines day one for the epoch for all date functions as Jan 1, 1900, and millisecond one for the epoch for all time functions is midnight, 00:00:00, Greenwich Mean Time (GMT). This definition means that negative time values can be returned to users in time zones east of GMT.

Date formats

A *date format* is a shorthand specification of how a date appears. It consists of various punctuation marks and symbols that represent the formatting that the date must use. The following table lists examples of date formats.

Date format	Example
MM/DD/YY	11/11/78
DD/MM/YY	25/07/85
MMMM DD, YYYY	March 10, 1964

The format of dates is governed by an ISO standard. Each country or region specifies its own date formats. The four general categories of date formats are short, medium, long, and full. The following table contains examples of different date formats from different locales for each of the categories.

Locale identifier and description	Date format (Category)	Example
en_GB English (United Kingdom)	DD/MM/YY (Short)	08/12/92 08/04/05
fr_CA French (Canada)	YY-MM-DD (Medium)	92-08-18
de_DE German (Germany)	D. MMMM YYYY (Long)	17. Juni 1989
fr_FR French (France)	EEEE, 'le ' D MMMM YYYY (Full)	Lundi, le 29 Octobre, 1990

Time formats

A *time format* is a shorthand specification to format a time. It consists of punctuations, literals, and pattern symbols. The following table lists examples of time formats.

Time format	Example
h:MM A	7:15 PM
HH:MM:SS	21:35:26
HH:MM:SS 'o'clock' A Z	14:20:10 o'clock PM EDT

Time formats are governed by an ISO standard. Each nation specifies the form of its default, short, medium, long, and full-time formats. The locale identifies the format of times that conform to the standards of that nation.

The following table contains some examples of different date formats from different locales for each of the categories.

Locale identifier and description	Time format (Category)	Example
en_GB English (United Kingdom)	HH:MM (Short)	14:13
fr_CA French (Canada)	HH:MM:SS (Medium)	12:15:50

Locale identifier and description	Time format (Category)	Example
de_DE German (Germany)	HH:MM:SS z (Long)	14:13:13 -0400
fr_FR French (France)	HH ' h ' MM Z (Full)	14 h 13 GMT-04:00

Date and time picture formats

The following symbols must be used to create date and time patterns for date/time fields. Certain date symbols are only used in Chinese, Japanese, and Korean locales. These symbols are also specified below.

NOTE: The comma (,), dash (-), colon (:), slash (/), period (.), and space () are treated as literal values and can be included anywhere in a pattern. To include a phrase in a pattern, delimit the text string with single quotation marks ('). For example, 'Your payment is due no later than' MM-DD-YY can be specified as the display pattern.

Date symbol	Description	Formatted value for English (USA) locale where the locale-sensitive input value is 1/1/08 (which is January 1, 2008)
D	1 or 2 digit (1-31) day of the month	1
DD	Zero-padded 2 digit (01-31) day of the month	01
J	1, 2, or 3 digit (1-366) day of the year	1
JJJ	Zero-padded, three-digit (001-366) day of the year	001
M	One- or two-digit (1-12) month of the year	1
MM	Zero-padded, two-digit (01-12) month of the year	01
MMM	Abbreviated month name	Jan
MMMM	Full month name	January
E	One-digit (1-7) day of the week, where (1=Sunday)	3 (because January 1, 2008 is a Tuesday)

Date symbol	Description	Formatted value for English (USA) locale where the locale-sensitive input value is 1/1/08 (which is January 1, 2008)
EEE	Abbreviated weekday name	Tue (because January 1, 2008 is a Tuesday)
EEEE	Full weekday name	Tuesday (because January 1, 2008 is a Tuesday)
YY	Two-digit year, where numbers less than 30 are considered to fall after the year 2000 and numbers 30 and higher are considered to occur before 2000. For example, 00=2000, 29=2029, 30=1930, and 99=1999	08
YYYY	Four-digit year	2008
G	Era name (BC or AD)	AD
w	One-digit (0-5) week of the month, where week 1 is the earliest set of four contiguous days ending on a Saturday	1
WW	Two-digit (01-53) ISO-8601 week of the year, where week 1 is the week containing January 4	01

Several additional date patterns are available for specifying date patterns in Chinese, Japanese, and Korean locales.

Japanese eras can be represented by several different symbols. The final four era symbols provide alternative symbols to represent Japanese eras.

CJK date symbol	Description
DDD	The locale's ideographic numeric valued day of the month
DDDD	The locale's tens rule ideographic numeric valued day of the month
YYY	The locale's ideographic numeric valued year
YYYYY	The locale's tens rule ideographic numeric valued year
g	The locale's alternate era name. For the current Japanese era, Heisei, this pattern displays the ASCII letter H (U+48)

CJK date symbol	Description
gg	The locale's alternate era name. For the current Japanese era, this pattern displays the ideograph that is represented by the Unicode symbol (U+5E73)
ggg	The locale's alternate era name. For the current Japanese era, this pattern displays the ideographs that are represented by the Unicode symbols (U+5E73 U+6210)
g	The locale's alternate era name. For the current Japanese era, this pattern displays the full width letter H (U+FF28)
g g	The locale's alternate era name. For the current Japanese era, this pattern displays the ideograph that is represented by the Unicode symbol (U+337B)

Time symbol	Description	Locale-sensitive input value	Formatted value for English (USA) locale
h	One- or two-digit (1-12) hour of the day (AM/PM)	12:08 AM or 2:08 PM	12 or 2
hh	Zero-padded 2 digit (01-12) hour of the day (AM/PM)	12:08 AM or 2:08 PM	12 or 02
k	One- or two-digit (0-11) hour of the day (AM/PM)	12:08 AM or 2:08 PM	0 or 2
kk	Two-digit (00-11) hour of the day (AM/PM)	12:08 AM or 2:08 PM	00 or 02
H	One- or two-digit (0-23) hour of the day	12:08 AM or 2:08 PM	0 or 14
HH	Zero-padded, two-digit (00-23) hour of the day	12:08 AM or 2:08 PM	00 or 14
K	One- or two-digit (1-24) hour of the day	12:08 AM or 2:08 PM	24 or 14
KK	Zero-padded, two-digit (01-24) hour of the day	12:08 AM or 2:08 PM	24 or 14
M	One- or two-digit (0-59) minute of the hour <i>NOTE: You must use this symbol with an hour symbol.</i>	2:08 PM	8

Time symbol	Description	Locale-sensitive input value	Formatted value for English (USA) locale
MM	Zero-padded, two-digit (00-59) minute of the hour <i>NOTE: You must use this symbol with an hour symbol.</i>	2:08 PM	08
S	One- or two-digit (0-59) second of the minute <i>NOTE: You must use this symbol with an hour and minute symbol.</i>	2:08:09 PM	9
SS	Zero-padded, two-digit (00-59) second of the minute <i>NOTE: You must use this symbol with an hour and minute symbol.</i>	2:08:09 PM	09
FFF	Three- digit (000-999) thousandth of the second <i>NOTE: You must use this symbol with an hour, minute, and seconds symbol.</i>	2:08:09 PM	09
A	The part of the day that is from midnight to noon (AM) or from noon to midnight (PM)	2:08:09 PM	PM
z	ISO-8601 time-zone format (for example, Z, +0500, -0030, -01, +0100) <i>NOTE: You must use this symbol with an hour symbol.</i>	2:08:09 PM	-0400
zz	Alternative ISO-8601 time-zone format (for example, Z, +05:00, -00:30, -01, +01:00) <i>NOTE: You must use this symbol with an hour symbol.</i>	2:08:09 PM	-04:00
Z	Abbreviated time-zone name (for example, GMT, GMT+05:00, GMT-00:30, EST, PDT) <i>NOTE: You must use this symbol with an hour symbol.</i>	2:08:09 PM	EDT

Reserved symbols

The following symbols have special meanings and cannot be used as literal text.

Sym bol	Description
?	When submitted, the symbol matches any one character. When merged for display, it becomes a space.
*	When submitted, the symbol matches 0 or Unicode white space characters. When merged for display, it becomes a space.
+	When submitted, the symbol matches one or more Unicode white space characters. When merged for display, it becomes a space.

Locales

For a list of supported languages, see [Locales](#) topic in the Using Designer guide.

For a list of supported languages, see [Locales](#) topic in the Using Designer guide.

Date

Returns the current system date as the number of days since the epoch.

Syntax

`Date ()`

Parameters

None

Examples

The following expression is an example of using the `Date` function:

Expression	Returns
<code>Date ()</code>	37875(the number of days from the epoch to September 12, 2003)

Date2Num

Returns the number of days since the epoch, given a date string.

Syntax

```
Date2Num(d [, f [, k ]])
```

Parameters

Parameter	Description
<i>d</i>	A date string in the format supplied by <i>f</i> that also conforms to the locale given by <i>k</i> .
<i>f</i> (optional)	A date format string. If <i>f</i> is omitted, the default date format <code>MMM D, YYYY</code> is used.
<i>k</i> (optional)	A locale identifier string that conforms to the locale naming standards. If <i>k</i> is omitted (or is invalid), the ambient locale is used.

The function returns a value of 0 if any of the following conditions are true:

- The format of the given date does not match the format specified in the function.
- Either the locale or date format supplied in the function is invalid.

Insufficient information is provided to determine a unique day since the epoch (that is, any information regarding the date is missing or incomplete).

Examples

The following expressions are examples of using the `Date2Num` function:

Expression	Returns
<code>Date2Num("Mar 15, 1996")</code>	35138
<code>Date2Num("1/1/1900", "D/M/YYYY")</code>	1
<code>Date2Num("03/15/96", "MM/DD/YY")</code>	35138

Expression	Returns
Date2Num("Aug 1, 1996", "MMM D, YYYY")	35277
Date2Num("96-08-20", "YY-MM-DD", "fr_FR")	35296
Date2Num("1/3/00", "D/M/YY") - Date2Num("1/2/00", "D/M/YY")	29

DateFmt

Returns a date format string, given a date format style.

Syntax

```
DateFmt([n [, k ]])
```

Parameters

Parameter	Description
n(optional)	<p>An integer identifying the locale-specific time format style as follows:</p> <ul style="list-style-type: none"> 1 (Short style) 2 (Medium style) 3 (Long style) 4 (Full style) <p>If n is omitted (or is invalid), the default style value 0 is used.</p>
k(optional)	<p>A locale identifier string that conforms to the locale naming standards. If k is omitted (or is invalid), the ambient locale is used.</p>

Examples

The following expressions are examples of using the DateFmt function:

Expression	Returns
DateFmt (1)	M/D/YY(if en_US locale is set)
DateFmt (2, "fr_CA")	YY-MM-DD
DateFmt (3, "de_DE")	D. MMMM YYYY
DateFmt (4, "fr_FR")	EEEE D' MMMM YYYY

IsoDate2Num

Returns the number of days since the epoch began, given a valid date string.

Syntax

`IsoDate2Num (d)`

Parameters

Parameter	Description
d	A valid date string.

Examples

The following expressions are examples of using the `IsoDate2Num` function:

Expression	Returns
<code>IsoDate2Num ("1900")</code>	1
<code>IsoDate2Num ("1900-01")</code>	1
<code>IsoDate2Num ("1900-01-01")</code>	1
<code>IsoDate2Num ("19960315T20:20:20")</code>	35138

Expression	Returns
<code>IsoDate2Num("2000-03-01") - IsoDate2Num("20000201")</code>	29

IsoTime2Num

Returns the number of milliseconds since the epoch, given a valid time string.

Syntax

```
IsoTime2Num(d)
```

Parameters

Parameter	Description
d	A valid time string.

Examples

The following expressions are examples of using the `IsoTime2Num` function:

Expression	Returns
<code>IsoTime2Num("00:00:00Z")</code>	1, for a user in the Eastern Time (ET) zone.
<code>IsoTime2Num("13")</code>	64800001, for a user located in Boston, U.S.
<code>IsoTime2Num("13:13:13")</code>	76393001, for a user located in California.
<code>IsoTime2Num("19111111T131313+01")</code>	43993001, for a user located in the Eastern Time (ET) zone.

LocalDateFmt

Returns a localized date format string, given a date format style.

Syntax

```
LocalDateFmt([n [, k ]])
```

Parameters

Parameter	Description
n(optional)	An integer identifying the locale-specific date format style as follows: <ul style="list-style-type: none">• 1 (Short style)• 2 (Medium style)• 3 (Long style)• 4 (Full style) If n is omitted (or is invalid), the default style value 0 is used.
k(optional)	A locale identifier string that conforms to the locale naming standards. If k is omitted (or is invalid), the ambient locale is used.

Examples

The following expressions are examples of the `LocalDateFmt` function:

Expression	Returns
<code>LocalDateFmt(1, "de_DE")</code>	<code>tt.MM.uu</code>
<code>LocalDateFmt(2, "fr_CA")</code>	<code>aa-MM-jj</code>
<code>LocalDateFmt(3, "de_CH")</code>	<code>t. MMMM jjjj</code>
<code>LocalDateFmt(4, "fr_FR")</code>	<code>EEEE j MMMM aaaa</code>

LocalTimeFmt

Returns a localized time format string, given a time format style.

Syntax

```
LocalTimeFmt([n [, k ]])
```

Parameters

Parameter	Description
n(Optional)	An integer identifying the locale-specific time format style as follows: <ul style="list-style-type: none">• 1 (Short style)• 2 (Medium style)• 3 (Long style)• 4 (Full style) If n is omitted (or is invalid), the default style value 0 is used.
k(Optional)	A locale identifier string that conforms to the locale naming standards. If k is omitted (or is invalid), the ambient locale is used.

Examples

The following expressions are examples of using the `LocalTimeFmt` function:

Expression	Returns
<code>LocalTimeFmt(1, "de_DE")</code>	<code>HH:mm</code>
<code>LocalTimeFmt(2, "fr_CA")</code>	<code>HH:mm:ss</code>
<code>LocalTimeFmt(3, "de_CH")</code>	<code>HH:mm:ss z</code>
<code>LocalTimeFmt(4, "fr_FR")</code>	<code>HH' h 'mm z</code>

Num2Date

Returns a date string, given a number of days since the epoch.

Syntax

```
Num2Date(n [, f [, k ]])
```

Parameters

Parameter	Description
<i>n</i>	An integer representing the number of days. If <i>n</i> is invalid, the function returns an error.
<i>f</i> (Optional))	A date format string. If you do not include a value for <i>f</i> , the function uses the default date format <code>MMM D, YYYY</code> .
<i>k</i> (Optional))	A locale identifier string that conforms to the locale naming standards. If you do not include a value for <i>k</i> , or if <i>k</i> is invalid, the function uses the ambient locale.

The function returns a value of 0 if any of the following conditions are true:

- The format of the given date does not match the format specified in the function.
- Either the locale or date format supplied in the function is invalid.

Insufficient information is provided to determine a unique day since the epoch (that is, any information regarding the date is missing or incomplete).

Examples

The following expressions are examples of using the `Num2Date` function:

Expression	Returns
<code>Num2Date(1, "DD/MM/YYYY")</code>	01/01/1900
<code>Num2Date(35139, "DD-MMM-YYYY", "de_DE")</code>	16-Mrz-1996

Expression	Returns
<code>Num2Date(Date2Num("Mar 15, 2000") - Date2Num("98-03-15", "YY-MM-DD", "fr_CA"))</code>	Jan 1, 1902

Num2GMTTime

Returns a GMT time string, given a number of milliseconds from the epoch.

Syntax

```
Num2GMTTime(n [, f [, k ]])
```

Parameters

Parameter	Description
<i>n</i>	An integer representing the number of milliseconds. If <i>n</i> is invalid, the function returns an error.
<i>f</i> (Optional)	A time format string. If you do not include a value for <i>f</i> , the function uses the default time format <code>H:MM:SS A</code> .
<i>k</i> (Optional)	A locale identifier string that conforms to the locale naming standards. If you do not include a value for <i>k</i> , or if <i>k</i> is invalid, the function uses the ambient locale.

The function returns a value of 0 if any of the following conditions are true:

- The format of the given time does not match the format specified in the function.
- Either the locale or time format supplied in the function is invalid.

Insufficient information is provided to determine a unique time since the epoch (that is, any information regarding the time is missing or incomplete).

Examples

The following expressions illustrate using the `Num2GMTTime` function:

Expression	Returns
<code>Num2GMTTime(1, "HH:MM:SS")</code>	<code>00:00:00</code>
<code>Num2GMTTime(65593001, "HH:MM:SS Z")</code>	<code>18:13:13 GMT</code>
<code>Num2GMTTime(43993001, TimeFmt(4, "de_DE"), "de_DE")</code>	<code>12.13 Uhr GMT</code>

Num2Time

Returns a time string, given a number of milliseconds from the epoch.

Syntax

```
Num2Time(n [, f [, k ]])
```

Parameters

Parameter	Description
<i>n</i>	An integer representing the number of milliseconds. If <i>n</i> is invalid, the function returns an error.
<i>f</i> (Optional)	A time format string. If you do not include a value for <i>f</i> , the function uses the default time format <code>H:MM:SS A</code> .
<i>k</i> (Optional)	A locale identifier string that conforms to the locale naming standards. If you do not include a value for <i>k</i> , or if <i>k</i> is invalid, the function uses the ambient locale.

The function returns a value of 0 if any of the following conditions are true:

- The format of the given time does not match the format specified in the function.
- Either the locale or time format supplied in the function is invalid.

Insufficient information is provided to determine a unique time since the epoch (that is, any information regarding the time is missing or incomplete).

Examples

The following expressions illustrate using the `Num2Time` function:

Expression	Returns
<code>Num2Time(1, "HH:MM:SS")</code>	00:00:00in Greenwich, England and09:00:00in Tokyo.
<code>Num2Time(65593001, "HH:MM:SS Z")</code>	13:13:13EST in Boston, U.S.
<code>Num2Time(65593001, "HH:MM:SS Z", "de_DE")</code>	13:13:13 GMT-05:00to a German-Swiss user in Boston, U.S.
<code>Num2Time(43993001, TimeFmt(4, "de_DE"), "de_DE")</code>	13.13 Uhr GMT+01:00to a user in Zurich, Austria.
<code>Num2Time(43993001, "HH:MM:SSzz")</code>	13:13+01:00to a user in Zurich, Austria.

Time

Returns the current system time as the number of milliseconds since the epoch.

Syntax

```
Time()
```

Parameters

None

Examples

The following expression is an example of using the `Time` function:

Expression	Returns
<code>Time()</code>	71533235at precisely 3:52:15 P.M. on September 15th, 2003 to a user in the Eastern Standard Time (EST) zone.

Time2Num

Returns the number of milliseconds since the epoch, given a time string.

Syntax

```
Time2Num(d [, f [, k ]])
```

Parameters

Parameter	Description
<i>d</i>	A time string in the format supplied by <i>f</i> that also conforms to the locale given by <i>k</i> .
<i>f</i> (Optional))	A time format string. If you do not include a value for <i>f</i> , the function uses the default time format <code>H:MM:SS A</code> .
<i>k</i> (Optional))	A locale identifier string that conforms to the locale naming standards. If you do not include a value for <i>k</i> , or if <i>k</i> is invalid, the function uses the ambient locale.

The function returns a value of 0 if any of the following conditions are true:

- The format of the given time does not match the format specified in the function.
- Either the locale or time format supplied in the function is invalid.

Insufficient information is provided to determine a unique time since the epoch (that is, any information regarding the time is missing or incomplete).

Examples

The following expressions illustrate using the `Time2Num` function:

Expression	Returns
<code>Time2Num("00:00:00 GMT", "HH:MM:SS Z")</code>	1
<code>Time2Num("1:13:13 PM")</code>	76393001to a user in California on Pacific Standard Time, and 76033001when that same user is on Pacific Daylight Savings Time.
<code>Time2Num("13:13:13", "HH:MM:SS") - Time2Num("13:13:13 GMT", "HH:MM:SS Z")) / (60 * 60 * 1000)</code>	8to a user in Vancouver and5to a user in Ottawa when on Standard Time. On Daylight Savings Time, the returned values are7and4, respectively.
<code>Time2Num("13:13:13 GMT", "HH:MM:SS Z", "fr_FR")</code>	47593001

TimeFmt

Returns a time format, given a time format style.

Syntax

```
TimeFmt([n [, k ]])
```

Parameters

Parameter	Description
n(Optional)	<p>An integer identifying the locale-specific time format style as follows:</p> <ul style="list-style-type: none"> 1 (Short style) 2 (Medium style) 3 (Long style) 4 (Full style) <p>If you do not include a value for n, or if n is invalid, the function uses the default style value.</p>

Parameter	Description
<code>k(Option al)</code>	A locale identifier string that conforms to the locale naming standards. If <code>k</code> is omitted (or is invalid), the ambient locale is used.

Examples

The following expressions are examples of using the `TimeFmt` function:

Expression	Returns
<code>TimeFmt(1)</code>	<code>h:MM A</code> (if <code>en_US</code> locale is set)
<code>TimeFmt(2, "fr_CA")</code>	<code>HH:MM:SS</code>
<code>TimeFmt(3, "fr_FR")</code>	<code>HH:MM:SS Z</code>
<code>TimeFmt(4, "de_DE")</code>	<code>H.MM' Uhr 'Z</code>

Financial Functions

Financial functions perform a variety of interest, principal, and evaluation calculations related to the financial sector.

Apr

Returns the annual percentage rate for a loan.

NOTE: Interest rate calculation methods differ from country to country. This function calculates an interest rate based on U.S. interest rate standards.

Syntax

`Apr(n1, n2, n3)`

Parameters

Parameter	Description
n1	A numeric value or expression representing the principal amount of the loan.
n2	A numeric value or expression representing the payment amount on the loan.
n3	A numeric value or expression representing the number of periods in the loan's duration.

If any parameter is null, the function returns `null`. If any parameter is negative or 0, the function returns an error.

NOTE: FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see *Numberliterals*.

Examples

The following expressions are examples that use the `Apr` function:

Expression	Returns
Apr(35000, 269.50, 360)	0.08515404566for a \$35,000 loan repaid at \$269.50 a month for 30 years.
Apr(210000 * 0.75, 850 + 110, 25 * 26)	0.07161332404
Apr(-20000, 250, 120)	Error
Apr(P_Value, Payment, Time)	This example uses variables in place of actual numeric values or expressions.

CTerm

Returns the number of periods needed for an investment earning a fixed, but compounded, interest rate to grow to a future value.

NOTE: Interest rate calculation methods differ from country to country. This function calculates an interest rate based on U.S. interest rate standards.

Syntax

CTerm(*n1*, *n2*, *n3*)

Parameters

Parameter	Description
n1	A numeric value or expression representing the interest rate per period.
n2	A numeric value or expression representing the future value of the investment.
n3	A numeric value or expression representing the amount of the initial investment.

If any parameter is null, the function returns null. If any parameter is negative or 0, the function returns an error.

NOTE: FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see *Numberliterals*.

Examples

The following expressions are examples that use the CTerm function:

Expression	Returns
CTerm(0.02, 1000, 100)	116.2767474515
CTerm(0.10, 500000, 12000)	39.13224648502
CTerm(0.0275 + 0.0025, 1000000, 55000 * 0.10)	176.02226044975
CTerm(Int_Rate, Target_Amount, P_Value)	This example uses variables in place of actual numeric values or expressions.

FV

Returns the future value of consistent payment amounts made at regular intervals at a constant interest rate.

NOTE: Interest rate calculation methods differ from country to country. This function calculates an interest rate based on U.S. interest rate standards.

Syntax

`FV(n1, n2, n3)`

Parameters

Parameter	Description
n1	A numeric value or expression representing the payment amount.
n2	A numeric value or expression representing the interest per period of the investment.

Parameter	Description
n3	A numeric value or expression representing the total number of payment periods.

The function returns an error if either of the following conditions are true:

- Either of n1 or n3 are negative or 0.
- n2 is negative.

If any of the parameters are null, the function returns null.

NOTE: FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see Numberliterals.

Examples

The following expressions are examples of the FV function:

Expression	Returns
<code>FV(400, 0.10 / 12, 30 * 12)</code>	904195.16991842445. This is the value, after 30 years, of a \$400 a month investment growing at 10% annually.
<code>FV(1000, 0.075 / 4, 10 * 4)</code>	58791.96145535981. This is the value, after 10 years, of a \$1000 a month investment growing at 7.5% a quarter.
<code>FV(Payment[0], Int_Rate / 4, Time)</code>	This example uses variables in place of actual numeric values or expressions.

IPmt

Returns the amount of interest paid on a loan over a set period of time.

NOTE: Interest rate calculation methods differ from country to country. This function calculates an interest rate based on U.S. interest rate standards.

Syntax

`IPmt(n1, n2, n3, n4, n5)`

Parameters

Parameter	Description
n1	A numeric value or expression representing the principal amount of the loan.
n2	A numeric value or expression representing the annual interest rate of the investment.
n3	A numeric value or expression representing the monthly payment amount.
n4	A numeric value or expression representing the first month in which a payment will be made.
n5	A numeric value or expression representing the number of months for which to calculate.

The function returns an error if either of the following conditions are true:

- n1, n2, or n3 are negative or 0.
- Either n4 or n5 are negative.

If any parameter is null, the function returns null. If the payment amount (n3) is less than the monthly interest load, the function returns 0.

NOTE: FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see Numberliterals.

Examples

The following expressions are examples that use the `IPmt` function:

Expression	Returns
<code>IPmt(30000, 0.085, 295.50, 7, 3)</code>	624.8839283142. The amount of interest repaid on a \$30000 loan at 8.5% for the three months between the seventh month and the tenth month of the loan's term.
<code>IPmt(160000, 0.0475, 980, 24, 12)</code>	7103.80833569485. The amount of interest repaid during the third year of the loan.

Expression	Returns
<code>IPmt(15000, 0.065, 65.50, 15, 1)</code>	0, because the monthly payment is less than the interest the loan accrues during the month.

NPV

Returns the net present value of an investment based on a discount rate and a series of periodic future cash flows.

NOTE: Interest rate calculation methods differ from country to country. This function calculates an interest rate based on U.S. interest rate standards.

Syntax

`NPV(n1, n2 [, ...])`

Parameters

Parameter	Description
n1	A numeric value or expression representing the discount rate over a single period.
n2	A numeric value or expression representing a cash flow value, which must occur at the end of a period. It is important that the values specified in n2 and beyond are in the correct sequence.

The function returns an error if n1 is negative or 0. If any of the parameters are null, the function returns null.

NOTE: FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [Number literals](#).

Examples

The following expressions are examples that use the NPV function:

Expression	Returns
NPV(0.065, 5000)	4694.83568075117, which is the net present value of an investment earning 6.5% per year that will generate \$5000.
NPV(0.10, 500, 1500, 4000, 10000)	11529.60863329007, which is the net present value of an investment earning 10% a year that will generate \$500, \$1500, \$4000, and \$10,000 in each of the next four years.
NPV(0.0275 / 12, 50, 60, 40, 100, 25)	273.14193838457, which is the net present value of an investment earning 2.75% year that will generate \$50, \$60, \$40, \$100, and \$25 in each of the next five months.

Pmt

Returns the payment for a loan based on constant payments and a constant interest rate.

NOTE: Interest rate calculation methods differ from country to country. This function calculates an interest rate based on U.S. interest rate standards.

Syntax

`Pmt(n1, n2, n3)`

Parameters

Parameter	Description
n1	A numeric value or expression representing the principal amount of the loan.
n2	A numeric value or expression representing the interest rate per period of the investment.
n3	A numeric value or expression representing the total number of payment periods.

The function returns an error if any parameter is negative or 0. If any parameter is null, the function returns null.

NOTE: FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see *Numberliterals*.

Examples

The following expressions are examples that use the `Pmt` function:

Expression	Returns
<code>Pmt(150000, 0.0475 / 12, 25 * 12)</code>	855.17604207164, which is the monthly payment on a \$150,000 loan at 4.75% annual interest, repayable over 25 years.
<code>Pmt(25000, 0.085, 12)</code>	3403.82145169876, which is the annual payment on a \$25,000 loan at 8.5% annual interest, repayable over 12 years.

PPmt

Returns the amount of principal paid on a loan over a period of time.

NOTE: Interest rate calculation methods differ from country to country. This function calculates an interest rate based on US interest rate standards.

Syntax

`PPmt(n1, n2, n3, n4, n5)`

Parameters

Parameter	Description
n1	A numeric value or expression representing the principal amount of the loan.
n2	A numeric value or expression representing the annual interest rate.
n3	A numeric value or expression representing the amount of the monthly payment.

Parameter	Description
n4	A numeric value or expression representing the first month in which a payment will be made.
n5	A numeric value or expression representing the number of months for which to calculate.

The function returns an error if either of the following conditions are true:

- n1, n2, or n3 are negative or 0.
- Either n4 or n5 is negative.

If any parameter is null, the function returns null. If the payment amount (n3) is less than the monthly interest load, the function returns 0.

***NOTE:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see Numberliterals.*

Examples

The following expressions are examples that use the PPmt function:

Expression	Returns
PPmt (30000, 0.085, 295.50, 7, 3)	261.6160716858, which is the amount of principal repaid on a \$30,000 loan at 8.5% for the three months between the seventh month and the tenth month of the loan's term.
PPmt (160000, 0.0475, 980, 24, 12)	4656.19166430515, which is the amount of principal repaid during the third year of the loan.
PPmt (15000, 0.065, 65.50, 15, 1)	0, because in this case the monthly payment is less than the interest the loan accrues during the month, therefore, no part of the principal is repaid.

PV

Returns the present value of an investment of periodic constant payments at a constant interest rate.

***NOTE:** Interest rate calculation methods differ from country to country. This function calculates an interest rate based on U.S. interest rate standards.*

Syntax

`PV(n1, n2, n3)`

Parameters

Parameter	Description
n1	A numeric value or expression representing the payment amount.
n2	A numeric value or expression representing the interest per period of the investment.
n3	A numeric value or expression representing the total number of payment periods.

The function returns an error if either n1 or n3 is negative or 0. If any parameter is null, the function returns null.

NOTE: FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see *Numberliterals*.

Examples

The following expressions are examples that use the PV function:

Expression	Returns
<code>PV(400, 0.10 / 12, 30 * 12)</code>	45580.32799074439. This is the value after 30 years, of a \$400 a month investment growing at 10% annually.
<code>PV(1000, 0.075 / 4, 10 * 4)</code>	58791.96145535981. This is the value after ten years of a \$1000 a month investment growing at 7.5% a quarter.
<code>PV(Payment[0], Int_Rate / 4, Time)</code>	This example uses variables in place of actual numeric values or expressions.

Rate

Returns the compound interest rate per period required for an investment to grow from present to future value in a given period.

NOTE: Interest rate calculation methods differ from country to country. This function calculates an interest rate based on U.S. interest rate standards.

Syntax

`Rate(n1, n2, n3)`

Parameters

Parameter	Description
n1	A numeric value or expression representing the future value of the investment.
n2	A numeric value or expression representing the present value of the investment.
n3	A numeric value or expression representing the total number of investment periods.

The function returns an error if any parameter is negative or 0. If any parameter is null, the function returns null.

NOTE: FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see *Numberliterals*.

Examples

The following expressions are examples that use the `Rate` function:

Expression	Returns
<code>Rate(12000, 8000, 5)</code>	0.0844717712(or 8.45%), which is the interest rate per period needed for an \$8000 present value to grow to \$12,000 in five periods.

Expression	Returns
Rate(10000, 0.25 * 5000, 4 * 12)	0.04427378243(or 4.43%), which is the interest rate per month needed for the present value to grow to \$10,000 in four years.
Rate(Target_Value, Pres_Value[*], Term * 12)	This example uses variables in place of actual numeric values or expressions.

Term

Returns the number of periods needed to reach a given future value from periodic constant payments into an interest bearing account.

NOTE: Interest rate calculation methods differ from country to country. This function calculates an interest rate based on U.S. interest rate standards.

Syntax

Term(*n1*, *n2*, *n3*)

Parameters

Parameter	Description
n1	A numeric value or expression representing the payment amount made at the end of each period.
n2	A numeric value or expression representing the interest rate per period of the investment.
n3	A numeric value or expression representing the future value of the investment.

The function returns an error if any parameter is negative or 0. If any parameter is null, the function returns null.

NOTE: FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see *Numberliterals*.

Examples

The following expressions are examples that use the `Term` function:

Expression	Returns
<code>Term(475, .05, 1500)</code>	3.00477517728 (or roughly 3), which is the number of periods needed to grow a payment of \$475 into \$1500, with an interest rate of 5% per period.
<code>Term(2500, 0.0275 + 0.0025, 5000)</code>	1.97128786369, which is the number of periods needed to grow payments of \$2500 into \$5000, with an interest rate of 3% per period.
<code>Rate(Inv_Value[0], Int_Rate + 0.0050, Target_Value)</code>	This example uses variables in place of actual numeric values or expressions. In this case, the first occurrence of the variable <code>Inv_Value</code> is used as the payment amount, half a percentage point is added to the variable <code>Int_Rate</code> to use as the interest rate, and the variable <code>Target_Value</code> is used as the future value of the investment.

Logical Functions

Logical functions are useful for testing and/or analyzing information to obtain a true or false result.

Choose

Selects a value from a given set of parameters.

Syntax

```
Choose(n, s1 [, s2 ...])
```

Parameters

Parameter	Description
<i>n</i>	The position of the value you want to select within the set. If this value is not a whole number, the function rounds <i>n</i> down to the nearest whole value. The function returns an empty string if either of the following conditions is true: <ul style="list-style-type: none"><i>n</i> is less than 1.<i>n</i> is greater than the number of items in the set. If <i>n</i> is null, the function returns null.
<i>s1</i>	The first value in the set of values.
<i>s2</i> (Optional)	Additional values in the set.

Examples

The following expressions are examples that use the `Choose` function:

Expression	Returns
<code>Choose(3, "Taxes", "Price", "Person", "Teller")</code>	Person
<code>Choose(2, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)</code>	9
<code>Choose(Item_Num[0], Items[*])</code>	Returns the value within the setItems that corresponds to the position defined by the first occurrence of Item_Num.
<code>Choose(20/3, "A", "B", "C", "D", "E", "F", "G", "H")</code>	F

Exists

Determines whether the given parameter is a reference syntax to an existing object.

Syntax

`Exists(v)`

Parameters

Parameter	Description
v	A valid reference syntax expression. If v is not a reference syntax, the function returns false (0).

Examples

The following expressions are examples that use the `Exists` function:

Expression	Returns
<code>Exists (Item)</code>	True (1) if the object <code>Item</code> exists and false (0) otherwise.
<code>Exists ("hello world")</code>	False (0). The string is not a reference syntax.
<code>Exists (Invoice.Border .Edge[1].Color)</code>	True (1) if the object <code>Invoice</code> exists and has a <code>Border</code> property, which in turn has at least one <code>Edge</code> property, which in turn has a <code>Color</code> property. Otherwise, the function returns false (0).

HasValue

Determines whether the given parameter is a reference syntax with a non-null, non-empty, or non-blank value.

Syntax

`HasValue (v)`

Parameters

Parameter	Description
<code>v</code>	A valid reference syntax expression. If <code>v</code> is not a reference syntax, the function returns false (0).

Examples

The following expressions are examples that use the `HasValue` function.

Expression	Returns
<code>HasValue (2)</code>	True (1)
<code>HasValue (" ")</code>	False (0)

Expression	Returns
HasValue (Amount[*])	Error
HasValue (Amount[0])	Evaluates the first occurrence of Amount and returns true (1) if it is a non-null, non-empty, or non-blank value.

Oneof

Determines whether the given value is within a set.

Syntax

```
Oneof(s1, s2 [, s3 ...])
```

Parameters

Parameter	Description
s1	The position of the value you want to select within the set. If this value is not a whole number, the function rounds s1 down to the nearest whole value.
s2	The first value in the set of values.
s3(Optional)	Additional values in the set.

Examples

The following expressions are examples that use the Oneof function:

Expression	Returns
Oneof(3, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)	True (1)
Oneof("John", "Bill", "Gary", "Joan", "John", "Lisa")	True (1)

Expression	Returns
<code>Oneof(3, 1, 25)</code>	<code>False(0)</code>
<code>Oneof("loan", Fields[*])</code>	Verifies whether any occurrence of <code>Fields</code> has a value of <code>loan</code> .

Within

Determines whether the given value is within a given range.

Syntax

```
Within(s1, s2, s3)
```

Parameters

Parameter	Description
<code>s1</code>	<p>The value to test for.</p> <p>If <code>s1</code> is a number, the ordering comparison is numeric.</p> <p>If <code>s1</code> is not a number, the ordering comparison uses the collating sequence for the current locale.</p> <p>For more information, see Locales.</p> <p>For more information, see Locales.</p> <p>If <code>s1</code> is null, the function returns null.</p>
<code>s2</code>	The lower bound of the test range.
<code>s3</code>	The upper bound of the test range.

Examples

The following expressions are examples that use the `Within` function:

Expression	Returns
<code>Within("C", "A", "D")</code>	True (1)
<code>Within(1.5, 0, 2)</code>	True (1)
<code>Within(-1, 0, 2)</code>	False (0)
<code>Within(\$, 1, 10)</code>	True (1) if the current value is between 1 and 10.

Miscellaneous Functions

Functions in this section do not fit within any other particular function category and are useful in a variety of applications.

Eval

Returns the value of a given form calculation.

Syntax

`Eval(s)`

Parameters

Parameter	Description
s	<p>A valid string representing an expression or list of expressions. TheEvalfunction cannot refer to user-defined variables and functions. For example:</p> <pre>var s = "var t = concat(s, ""hello"") " eval(s)</pre> <p>In this case, theEvalfunction does not recognizes, and so returns an error. Any subsequent functions that make reference to the variablesalso fail.</p>

Examples

The following expressions are examples that use the Eval function:

Expression	Returns
<code>eval("10*3+5*4")</code>	50
<code>eval("hello")</code>	error

Null

Returns the null value. The null value means no value.

Definition

`Null()`

Parameters

None

Examples

The following expressions are examples that use the `Null` function:

Expression	Returns
<code>Null()</code>	<code>null</code>
<code>Null() + 5</code>	5
<code>Quantity = Null()</code>	Assigns <code>null</code> to the object <code>Quantity</code> .
<code>Concat("ABC", Null(), "DEF")</code>	ABCDEF See also <code>Concat</code> .

Ref

Returns a reference to an existing object.

Definition

`Ref(v)`

Parameters

Parameters	Description
<code>v</code>	A valid string representing a reference syntax, property, method, or function. If the given parameter is null, the function returns the null reference. For all other given parameters, the function generates an error exception.

Examples

The following expressions are examples that use the `Ref` function:

Expressions	Returns
<code>Ref("10*3+5*4")</code>	<code>10*3+5*4</code>
<code>Ref("hello")</code>	<code>hello</code>

UnitType

Returns the units of a unitspan. A unitspan is a string consisting of a number followed by a unit name.

Syntax

```
UnitType(s)
```

Parameters

Parameter	Description
s	<p>A valid string containing a numeric value and a valid unit of measurement (unitspan). Recognized units of measurement are:</p> <ul style="list-style-type: none">• in, inches• mm, millimeters• cm, centimeters• pt, points• pc, picas• mp, millipoints <p>If s is invalid, the function returns in.</p>

Examples

The following expressions are examples that use the `UnitType` function:

Expression	Results
<code>UnitType("36 in")</code>	in
<code>UnitType("2.54centimeters")</code>	cm
<code>UnitType("picas")</code>	pc
<code>UnitType("2.cm")</code>	cm
<code>UnitType("2.zero cm")</code>	in
<code>UnitType("kilometers")</code>	in
<code>UnitType(Size[0])</code>	Returns the measurement value of the first occurrence of <code>Size</code> .

UnitValue

Returns the numerical value of a measurement with its associated unitspan, after an optional unit conversion. A unitspan is a string consisting of a number followed by a valid unit of measurement.

Syntax

```
UnitValue(s1 [, s2 ])
```

Parameters

Parameter s	Description
s1	A valid string containing a numeric value and a valid unit of measurement (unitspan). Recognized units of measurement are: <ul style="list-style-type: none">• in, inches• mm, millimeters• cm, centimeters• pt, picas, points• mp, millipoints
s2(optional)	A string containing a valid unit of measurement. The function converts the unitspan specified in s1 to this new unit of measurement. If you do not include a value for s2, the function uses the unit of measurement specified in s1. If s2 is invalid, the function converts s1 into inches.

Examples

The following expressions are examples that use the UnitValue function:

Expression	Returns
UnitValue("2in")	2
UnitValue("2in", "cm")	5.08

Expression	Returns
UnitValue("6", "pt")	432
UnitValue("A", "cm")	0
UnitValue(Size[2], "mp")	Returns the measurement value of the third occurrence ofSizeconverted into millipoints.
UnitValue("5.08cm", "kilograms")	2

String Functions

String functions deal with the manipulation, evaluation, and creation of string values.

At

Locates the starting character position of a string within another string.

Syntax

```
At(s1, s2)
```

Parameters

Parameter	Description
<i>s1</i>	The source string.
<i>s2</i>	The search string. If <i>s2</i> is not a part of <i>s1</i> , the function returns 0. If <i>s2</i> is empty, the function returns 1.

Examples

The following expressions are examples that use the `At` function:

Expression	Returns
<code>At("ABCDEFGH", "AB")</code>	1
<code>At("ABCDEFGH", "F")</code>	6
<code>At(23412931298471, 29)</code>	5, the first occurrence of 29 within the source string.
<code>At(Ltrim(Cust_Info[0]), "555")</code>	The location of the string 555 within the first occurrence of <code>Cust_Info</code> . See also <code>Ltrim</code> .

Concat

Returns the concatenation of two or more strings.

Syntax

```
Concat(s1 [, s2 ...])
```

Parameters

Parameter	Description
<i>s1</i>	The first string in the set.
<i>s2</i> (Optional)	Additional strings to append to the set.

Examples

The following expressions are examples that use the `Concat` function:

Expression	Returns
<code>Concat("ABC", "DEF")</code>	ABCDEF
<code>Concat("Tony", Space(1), "Blue")</code>	Tony Blue See also <code>Space</code> .
<code>Concat("You owe ", WordNum(1154.67, 2), ".")</code>	You owe One Thousand One Hundred Fifty-four Dollars And Sixty-seven Cents. See also <code>WordNum</code> .

Decode

Returns the decoded version of a given string.

Syntax

```
Decode(s1 [, s2 ])
```

Parameters

Parameter	Description
<i>s1</i>	The string to decode.
<i>s2</i> (Optional)	A string identifying the type of decoding to perform. The following strings are valid decoding strings: <ul style="list-style-type: none">• url (URL decoding)• html (HTML decoding)• xml (XML decoding) If you do not include a value for <i>s2</i> , the function uses URL decoding.

Examples

The following expressions are examples that use the `Decode` function:

Expression	Returns
<code>Decode("&AElig;&Aacute;&Acirc;&Aacute;&Acirc;", "html")</code>	ÆÁÂÃ
<code>Decode("~!@#\$\$%^&*()_+ `{&q uot;}[] &lt;&gt;;? ,./; &apos;;:", "xml")</code>	~!@#\$\$%^&*()_+ `{" } [] <>?,./; ' ;

Encode

Returns the encoded version of a given string.

Syntax

```
Encode(s1 [, s2 ])
```

Parameters

Parameter	Description
s1	The string to encode.
s2(Optional)	<p>A string identifying the type of encoding to perform. The following strings are valid encoding strings:</p> <ul style="list-style-type: none">• url (URL encoding)• html (HTML encoding)• xml (XML encoding) <p>If you do not include a value for s2, the function uses URL encoding.</p>

Examples

The following expressions are examples that use the `Encode` function:

Expression	Returns
<code>Encode("hello, world!", "url")</code>	<code>%22hello,%20world!%22</code>
<code>Encode("ÃÄÅÆ", "html")</code>	<code>&#xc1;&#Xc2;&#Xc3;&#xc4;&#xc5;&#xc6;</code>

Format

Formats the given data according to the specified picture format string.

Syntax

```
Format(s1, s2 [, s3 ...])
```

Parameters

Parameter	Description
s1	The picture format string, which may be a locale-sensitive date or time format. See Locales.
s2	<p>The source data to format.</p> <p>For date picture formats, the source data must be either an ISO date-time string or an ISO date string in one of two formats:</p> <ul style="list-style-type: none">• YYYY[MM[DD]]• YYYY[-MM[-DD]]• HH[MM[SS[.FFF][z]]]• HH[MM[SS[.FFF][+HH[MM]]]]• HH[MM[SS[.FFF][-HH[MM]]]]• HH[:MM[:SS[.FFF][z]]]• HH[:MM[:SS[.FFF][-HH[:MM]]]]• HH[:MM[:SS[.FFF][+HH[:MM]]]] <p>For time picture formats, the source data must be either an ISO date-time string or an ISO time string in one of the following formats:</p> <ul style="list-style-type: none">• HH[MM[SS[.FFF][z]]]• HH[MM[SS[.FFF][+HH[MM]]]]• HH[MM[SS[.FFF][-HH[MM]]]]• HH[:MM[:SS[.FFF][z]]]• HH[:MM[:SS[.FFF][-HH[:MM]]]]• HH[:MM[:SS[.FFF][+HH[:MM]]]] <p>For date-time picture formats, the source data must be an ISO date-time string.</p> <p>For numeric picture formats, the source data must be numeric.</p> <p>For text picture formats, the source data must be textual.</p> <p>For compound picture formats, the number of source data arguments must match the number of subelements in the picture.</p>
s3(Optional)	Additional source data to format.

Examples

The following expressions are examples that use the `Format` function:

Expression	Returns
<code>Format("MMM D, YYYY", "20020901")</code>	Sep 1, 2002
<code>Format("\$9,999,999.99", 1234567.89)</code>	\$1,234,567.89 in the U.S. and1 234 567,89Euros in France.

Left

Extracts a specified number of characters from a string, starting with the first character on the left.

Syntax

`Left(s, n)`

Parameters

Parameter	Description
<code>s</code>	The string to extract from.
<code>n</code>	The number of characters to extract. If the number of characters to extract is greater than the length of the string, the function returns the whole string. If the number of characters to extract is 0 or less, the function returns the empty string.

Examples

The following expressions are examples that use the `Left` function:

Expression	Returns
<code>Left ("ABCDEFGH", 3)</code>	ABC
<code>Left ("Tony Blue", 5)</code>	"Tony "
<code>Left (Telephone[0], 3)</code>	The first three characters of the first occurrence ofTelephone.
<code>Left (Rtrim (Last_Name), 3)</code>	The first three characters ofLast_Name. See also Rtrim.

Len

Returns the number of characters in a given string.

Syntax

`Len (s)`

Parameters

Parameter	Description
s	The string to examine.

Examples

The following expressions are examples that use the `Len` function:

Expression	Returns
<code>Len ("ABDCEFGH")</code>	8
<code>Len (4)</code>	1

Expression	Returns
<code>Len(Str(4.532, 6, 4))</code>	6 See also <code>Str</code> .
<code>Len(Amount[*])</code>	The number of characters in the first occurrence of <code>Amount</code> .

Lower

Converts all uppercase characters within a specified string to lowercase characters.

Syntax

```
Lower(s, [, k ])
```

Parameters

Parameter	Description
<code>s</code>	The string to convert.
<code>k(Optional)</code>	<p>A string representing a valid locale. If you do not include a value for <code>k</code>, the function uses the ambient locale. See Locales.</p> <p>This function only converts the Unicode characters U+41 through U+5A (of the ASCII character set) as well as the characters U+FF21 through U+FF3A (of the fullwidth character set)</p>

Examples

The following expressions are examples that use the `Lower` function:

Expression	Returns
<code>Lower("ABC")</code>	abc
<code>Lower("21 Main St.")</code>	21 main st.

Expression	Returns
<code>Lower(15)</code>	15
<code>Lower(Address[0])</code>	This example converts the first occurrence of <code>Address</code> to all lowercase letters.

Ltrim

Returns a string with all leading white space characters removed.

White space characters include the ASCII space, horizontal tab, line feed, vertical tab, form feed, carriage return, and the Unicode space characters (Unicode category Zs).

Syntax

```
Ltrim(s)
```

Parameters

Parameter	Description
<code>s</code>	The string to trim.

Examples

The following expressions are examples that use the `Ltrim` function:

Expression	Returns
<code>Ltrim(" ABCD")</code>	"ABCD"
<code>Ltrim(Rtrim(" Tony Blue "))</code>	"Tony Blue" See also <code>Rtrim</code> .
<code>Ltrim(Address[0])</code>	Removes any leading white space from the first occurrence of <code>Address</code> .

Parse

Analyzes the given data according to the given picture format.

Parsing data successfully results in one of the following values:

- Date picture format: An ISO date string of the form YYYY-MM-DD.
- Time picture format: An ISO time string of the form HH:MM:SS.
- Date-time picture format: An ISO date-time string of the form YYYY-MM-DDTHH:MM:SS.
- Numeric picture format: A number.
- Text pictures: Text.

Syntax

```
Parse(s1, s2 )
```

Parameters

Parameter	Description
s1	A valid date or time picture format string. For more information on date and time formats, see Dateand Time Functions.
s2	The string data to parse.

Examples

The following expressions are examples that use the `Parse` function:

Expression	Returns
<code>Parse("MMM D, YYYY", "Sep 1, 2002")</code>	2002-09-01
<code>Parse("\$9,999,999.99", "\$1,234,567.89")</code>	1234567.89in the U.S.

Replace

Replaces all occurrences of one string with another within a specified string.

Syntax

```
Replace(s1, s2 [, s3 ])
```

Parameters

Parameter	Description
<i>s1</i>	A source string.
<i>s2</i>	The string to replace.
<i>s3</i> (Optional)	The replacement string. If you do not include a value for <i>s3</i> , or if <i>s3</i> is null, the function uses an empty string.

Examples

The following expressions are examples that use the `Replace` function:

Expression	Returns
<code>Replace("Tony Blue", "Tony", "Chris")</code>	Chris Blue
<code>Replace("ABCDEFGH", "D")</code>	ABCEFGH
<code>Replace("ABCDEFGH", "d")</code>	ABCDEFGH
<code>Replace(Comments[0], "recieve", "receive")</code>	Correctly updates the spelling of the word <code>receive</code> in the first occurrence of <code>Comments</code> .

Right

Extracts a number of characters from a given string, beginning with the last character on the right.

Syntax

```
Right(s, n )
```

Parameters

Parameter	Description
s	The string to extract.
n	The number of characters to extract. If n is greater than the length of the string, the function returns the whole string. If n is 0 or less, the function returns an empty string.

Examples

The following expressions are examples that use the `Right` function:

Expression	Returns
<code>Right("ABCDEFGH", 3)</code>	FGH
<code>Right("Tony Blue", 5)</code>	" Blue"
<code>Right(Telephone[0], 7)</code>	The last seven characters of the first occurrence of Telephone.
<code>Right(Rtrim(CreditCard_Num), 4)</code>	The last four characters of CreditCard_Num. See also Rtrim.

Rtrim

Returns a string with all trailing white space characters removed.

White space characters include the ASCII space, horizontal tab, line feed, vertical tab, form feed, carriage return, and the Unicode space characters (Unicode category Zs).

Syntax

```
Rtrim(s )
```

Parameters

Parameter	Description
<i>s</i>	The string to trim.

Examples

The following expressions are examples that use the `Rtrim` function:

Expression	Returns
<code>Rtrim("ABCD ")</code>	<code>"ABCD"</code>
<code>Rtrim("Tony Blue ")</code>	<code>"Tony Blue"</code>
<code>Rtrim(Address[0])</code>	Removes any trailing white space from the first occurrence of <code>Address</code> .

Space

Returns a string consisting of a given number of blank spaces.

Syntax

```
Space(n )
```

Parameters

Parameter	Description
n	The number of blank spaces.

Examples

The following expressions are examples that use the `Space` function:

Expression	Returns
<code>Space (5)</code>	" "
<code>Space (Max (Amount [*]))</code>	A blank string with as many characters as the value of the largest occurrence of <code>Amount</code> . See also <code>Max</code> .
<code>Concat ("Tony", Space (1), "Blue")</code>	Tony Blue

Str

Converts a number to a character string. FormCalc formats the result to the specified width and rounds to the specified number of decimal places.

Syntax

```
Str(n1 [, n2 [, n3 ]])
```

Parameters

Parameter	Description
n1	The number to convert.

Parameter	Description
n2(Optional)	The maximum width of the string. If you do not include a value for n2, the function uses a value of 10 as the default width. If the resulting string is longer than n2, the function returns a string of * (asterisk) characters of the width specified by n2.
n3(Optional)	The number of digits to appear after the decimal point. If you do not include a value for n3, the function uses 0 as the default precision.

Examples

The following expressions are examples that use the `Str` function:

Expression	Returns
<code>Str(2.456)</code>	" 2"
<code>Str(4.532, 6, 4)</code>	4.5320
<code>Str(234.458, 4)</code>	" 234"
<code>Str(31.2345, 4, 2)</code>	****
<code>Str(Max(Amount[*]), 6, 2)</code>	Converts the largest occurrence of Amount to a six-character string with two decimal places. See also Max.

Stuff

Inserts a string into another string.

Syntax

```
Stuff(s1, n1, n2 [, s2 ])
```

Parameters

Parameter	Description
s1	The source string.
n1	The position in s1 to insert the new string s2. If n1 is less than one, the function assumes the first character position. If n1 is greater than length of s1, the function assumes the last character position.
n2	The number of characters to delete from string s1, starting at character position n1. If n2 is less than or equal to 0, the function assumes 0 characters.
s2(Optional)	The string to insert into s1. If you do not include a value for s2, the function uses the empty string.

Examples

The following expressions are examples that use the `Stuff` function:

Expression	Returns
<code>Stuff("TonyBlue", 5, 0, " ")</code>	Tony Blue
<code>Stuff("ABCDEFGH", 4, 2)</code>	ABCFGH
<code>Stuff(Address[0], Len(Address[0]), 0, "Street")</code>	This adds the word <code>Street</code> onto the end of the first occurrence of <code>Address</code> . See also <code>Len</code> .
<code>Stuff("members-list@myweb.com", 0, 0, "cc:")</code>	cc:members-list@myweb.com

Substr

Extracts a portion of a given string.

Syntax

`Substr(s1, n1, n2)`

Parameters

Parameter	Description
s1	The source string.
n1	The position in string s1 to start extracting. If n1 is less than one, the function assumes the first character position. If n1 is greater than length of s1, the function assumes the last character position.
n2	The number of characters to extract. If n2 is less than or equal to 0, FormCalc returns an empty string. If n1 + n2 is greater than the length of s1, the function returns the substring starting at position n1 to the end of s1.

Examples

The following expressions are examples that use the `Substr` function:

Expression	Returns
<code>Substr("ABCDEFGH", 3, 4)</code>	CDEF
<code>Substr(3214, 2, 1)</code>	2
<code>Substr>Last_Name[0], 1, 3)</code>	Returns the first three characters from the first occurrence of Last_Name.
<code>Substr("ABCDEFGH", 5, 0)</code>	" "
<code>Substr("21 Waterloo St.", 4, 5)</code>	Water

Uuid

Returns a Universally Unique Identifier (UUID) string to use as an identification method.

Syntax

```
Uuid([n ])
```

Parameters

Parameter	Description
n	<p>A number identifying the format of the UUID string. Valid numbers are:</p> <ul style="list-style-type: none">• 0 (default value): UUID string only contains hex octets.• 1: UUID string contains dash characters separating the sequences of hex octets at fixed positions. <p>If you do not include a value for n, the function uses the default value.</p>

Examples

The following expressions are examples of the `Uuid` function:

Expression	Returns
<code>Uuid()</code>	A value such as <code>3c3400001037be8996c400a0c9c86dd5</code>
<code>Uuid(0)</code>	A value such as <code>3c3400001037be8996c400a0c9c86dd5</code>
<code>Uuid(1)</code>	A value such as <code>1a3ac000-3dde-f352-96c4-00a0c9c86dd5</code>
<code>Uuid(7)</code>	A value such as <code>1a3ac000-3dde-f352-96c4-00a0c9c86dd5</code>

Upper

Converts all lowercase characters within a string to uppercase.

Syntax

```
Upper (s [, k ])
```

Parameters

Parameter	Description
s	The string to convert. See Locales.
k(Optional)	A string representing a valid locale. If you do not include a value for k, the ambient locale is used. This function only converts the Unicode characters U+61 through U+7A (of the ASCII character set) as well as the characters U+FF41 through U+FF5A (of the fullwidth character set).

Examples

The following expressions are examples that use the `Upper` function:

Expression	Returns
<code>Upper ("abc")</code>	ABC
<code>Upper ("21 Main St.")</code>	21 MAIN ST.
<code>Upper (15)</code>	15
<code>Upper (Address [0])</code>	This example converts the first occurrence of <code>Address</code> to all uppercase letters.

WordNum

Returns the English text equivalent of a given number.

Syntax

```
WordNum(n1 [, n2 [, k ]])
```

Parameters

Parameter	Description
<i>n1</i>	The number to convert. If any of the following statements is true, the function returns * (asterisk) characters to indicate an error: <ul style="list-style-type: none"><i>n1</i> is not a number.The integral value of <i>n1</i> is negative.The integral value of <i>n1</i> is greater than 922,337,203,685,477,550.
<i>n2</i> (Optional)	A number identifying the formatting option. Valid numbers are: <ul style="list-style-type: none">0 (default value): The number is converted into text representing the simple number.1: The number is converted into text representing the monetary value with no fractional digits.2: The number is converted into text representing the monetary value with fractional digits. If you do not include a value for <i>n2</i> , the function uses the default value (0).
<i>k</i> (Optional)	A string representing a valid locale. If you do not include a value for <i>k</i> , the function uses the ambient locale. See Locales . As of this release, it is not possible to specify a locale identifier other than English for this function.

Examples

The following expressions are examples that use the `WordNum` function.

Expression	Returns
<code>WordNum(123.45)</code>	One Hundred and Twenty-three Dollars

Expression	Returns
WordNum(123.45, 1)	One Hundred and Twenty-three Dollars
WordNum(1154.67, 2)	One Thousand One Hundred Fifty-four Dollars And Sixty-seven Cents
WordNum(43, 2)	Forty-three Dollars And Zero Cents
WordNum(Amount[0], 2)	This example uses the first occurrence of Amount as the conversion number.

URL Functions

URL functions deal with the sending and receiving of information, including content types and encoding data, to any accessible URL locations.

Get

Downloads the contents of the given URL.

NOTE: Adobe® Acrobat® and Adobe® Reader® cannot verify that the form is certified until after the `initialize` event initiates. To use the `Get` function on certified forms prior to the form rendering, use the `docReady` event.

Syntax

`Get (s)`

Parameters

Parameter	Description
s	The URL to download. If the function is unable to download the URL, it returns an error.

Examples

The following expressions are examples that use the `Get` function.

Expression	Returns
<code>Get ("http://www.myweb.com/data/mydata.xml")</code>	XML data taken from the specified file.
<code>Get ("ftp://ftp.gnu.org/gnu/GPL")</code>	The contents of the GNU Public License.

Expression	Returns
<code>Get("http://intranet?sql=SELECT+*+FROM+projects+FOR+XML+AUTO,+ELEMENTS")</code>	The results of an SQL query to the specified website.

Post

Posts the given data to the specified URL.

NOTE: Acrobat and Adobe Reader cannot verify that the form is certified until after the *initialize* event initiates. To use the *Post* function on certified forms prior to the form rendering, use the *docReady* event.

Syntax

```
Post(s1, s2 [, s3 [, s4 [, s5 ]]])
```

Parameters

Parameter	Description
s1	The URL to post to.
s2	The data to post. If the function cannot post the data, it returns an error.

Parameter	Description
<code>s3</code> (Optional)	<p>A string containing the content type of the data to post. Here are valid content types:</p> <ul style="list-style-type: none"> • <code>application/octet-stream</code> (default value) • <code>text/html</code> • <code>text/xml</code> • <code>text/plain</code> • <code>multipart/form-data</code> • <code>application/x-www-form-urlencoded</code> • Any other valid MIME type <p>If you do not include a value for <code>s3</code>, the function sets the content type to the default value. The application ensures that the data to post uses the correct format according to the specified content type.</p>
<code>s4</code> (Optional)	<p>A string containing the name of the code page used to encode the data. Here are valid code page names:</p> <ul style="list-style-type: none"> • <code>UTF-8</code> (default value) • <code>UTF-16</code> • <code>ISO-8859-1</code> • Any character encoding listed by the Internet Assigned Numbers Authority (IANA) <p>If you do not include a value for <code>s4</code>, the function sets the code page to the default value. The application ensures that encoding of the data to post matches the specified code page.</p>
<code>s5</code> (Optional)	<p>A string containing any additional HTTP headers to be included with the posting of the data.</p> <p>If you do not include a value for <code>s5</code>, the function does not include an additional HTTP header in the post.</p> <p>SOAP servers usually require a <code>SOAPAction</code> header when posting to them.</p>

Examples

The following expressions are examples that use the `Post` function:

Expression	Returns
<pre>Post("http://tools_build/scripts/jfecho.cgi", "user=joe&passwd=xxxxx&date=27/08/2002", "application/x-www-form-urlencoded")</pre>	Posts some URL encoded login data to a server and returns that server's acknowledgement page.
<pre>Post("http://www.nanonull.com/TimeService/TimeService.asmx/getLocalTime", "<?xml version='1.0' encoding='UTF-8'?><soap:Envelope><soap:Body><getLocalTime/></soap:Body></soap:Envelope>", "text/xml", "utf-8", "http://www.Nanonull.com/TimeService/getLocalTime")</pre>	Posts a SOAP request for the local time to some server, expecting an XML response back.

Put

Uploads the given data to the specified URL.

NOTE: Acrobat and Adobe Reader cannot verify that the form is certified until after the `initialize` event initiates. To use the `Put` function on certified forms prior to the form rendering, use the `docReady` event.

Syntax

```
Put(s1, s2 [, s3 ])
```

Parameters

Parameter	Description
s1	The URL to upload.
s2	The data to upload. If the function is unable to upload the data, it returns an error.

Parameter	Description
s3(Optional)	<p>A string containing the name of the code page used to encode the data. Here are valid code page names:</p> <ul style="list-style-type: none"> • UTF-8 (default value) • UTF-16 • ISO8859-1 • Any character encoding listed by the Internet Assigned Numbers Authority (IANA) <p>If you do not include a value for s3, the function sets the code page to the default value. The application ensures that encoding of the data to upload matches the specified code page.</p>

Examples

The following expression is an example that use the `Put` function:

Expression	Returns
<pre>Put ("ftp://www.example.com/pub/fubu.xml", "<?xml version='1.0' encoding='UTF-8'?><msg>hello world!</msg>")</pre>	<p>Nothing if the FTP server has permitted the user to upload some XML data to thepub/fubu.xmlfile. Otherwise, this function returns an error.</p>

NOTE: This example only works in the server environment and not in Acrobat or Adobe Reader. For forms displayed in Acrobat and Adobe Reader, use the HTTP, HTTPS, and FILE protocols.