DESIGNER FORMCALC USER REFERENCE

Contents

Chapter 1: About FormCalc

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.

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.

More Help topics

"Alphabetical Functions List" on page 8

Chapter 2: 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:

```
99999999999999999
```

rounds to the (nearest) value:

```
100000000000000000
```

This behavior can sometimes lead to surprising results. FormCalc provides a function, “Round” on page 16, 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 `0.0449999999999999`. 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\u006f\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
	&#x000D;
	&#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
Space	#x20
Vertical Tab	#xB

Chapter 3: 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" on page 11	Returns the absolute value of a numeric value or expression.	Arithmetic
"Apr" on page 31	Returns the annual percentage rate for a loan.	Financial
"At" on page 47	Locates the starting character position of a string within another string.	String
"Avg" on page 11	Evaluates a set of number values and/or expressions and returns the average of the non-null elements contained within that set.	Arithmetic
"Ceil" on page 12	Returns the whole number greater than or equal to a given number.	Arithmetic
"Choose" on page 39	Selects a value from a given set of parameters.	Logical
"Concat" on page 47	Returns the concatenation of two or more strings.	String
"Count" on page 12	Evaluates a set of values and/or expressions and returns the number of non-null elements contained within the set.	Arithmetic
"CTerm" on page 31	Returns the number of periods needed for an investment earning a fixed, but compounded, interest rate to grow to a future value.	Financial
"Date" on page 22	Returns the current system date as the number of days since the "Epoch" on page 18	Date and Time
"Date2Num" on page 22	Returns the number of days since the "Epoch" on page 18, given a date string.	Date and Time
"DateFmt" on page 23	Returns a date format string, given a date format style.	Date and Time
"Decode" on page 48	Returns the decoded version of a given string.	String
"Encode" on page 49	Returns the encoded version of a given string.	String
"Eval" on page 43	Returns the value of a given form calculation.	Miscellaneous
"Exists" on page 39	Determines whether the given parameter is a reference syntax to an existing object.	Logical
"Floor" on page 13	Returns the largest whole number that is less than or equal to the given value.	Arithmetic
"Format" on page 49	Formats the given data according to the specified picture format string.	String
"FV" on page 32	Returns the future value of consistent payment amounts made at regular intervals at a constant interest rate.	Financial
"Get" on page 61	Downloads the contents of the given URL.	URL
"HasValue" on page 40	Determines whether the given parameter is an accessor with a non-null, non-empty, or non-blank value.	Logical
"IPmt" on page 33	Returns the amount of interest paid on a loan over a set period of time.	Financial
"IsoDate2Num" on page 24	Returns the number of days since the "Epoch" on page 18, given an valid date string.	Date and Time
"IsoTime2Num" on page 24	Returns the number of milliseconds since the "Epoch" on page 18, given a valid time string.	Date and Time

Function	**Description**	**Type**
"Left" on page 51	Extracts a specified number of characters from a string, starting with the first character on the left.	String
"Len" on page 51	Returns the number of characters in a given string.	String
"LocalDateFmt" on page 25	Returns a localized date format string, given a date format style.	Date and Time
"LocalTimeFmt" on page 25	Returns a localized time format string, given a time format style.	Date and Time
"Lower" on page 52	Converts all uppercase characters within a specified string to lowercase characters.	String
"Ltrim" on page 52	Returns a string with all leading white space characters removed.	String
"Max" on page 14	Returns the maximum value of the non-null elements in the given set of numbers.	Arithmetic
"Min" on page 14	Returns the minimum value of the non-null elements of the given set of numbers.	Arithmetic
"Mod" on page 15	Returns the modulus of one number divided by another.	Arithmetic
"NPV" on page 34	Returns the net present value of an investment based on a discount rate and a series of periodic future cash flows.	Financial
"Null" on page 43	Returns the null value. The null value means no value.	Miscellaneous
"Num2Date" on page 26	Returns a date string, given a number of days since the "Epoch" on page 18.	Date and Time
"Num2GMTime" on page 27	Returns a GMT time string, given a number of milliseconds from the "Epoch" on page 18.	Date and Time
"Num2Time" on page 27	Returns a time string, given a number of milliseconds from the "Epoch" on page 18.	Date and Time
"Oneof" on page 40	Returns true (1) if a value is in a given set, and false (0) if it is not.	Logical
"Parse" on page 53	Analyzes the given data according to the given picture format.	String
"Pmt" on page 35	Returns the payment for a loan based on constant payments and a constant interest rate.	Financial
"Post" on page 61	Posts the given data to the specified URL.	URL
"PPmt" on page 35	Returns the amount of principal paid on a loan over a period of time.	Financial
"Put" on page 63	Uploads the given data to the specified URL.	URL
"PV" on page 36	Returns the present value of an investment of periodic constant payments at a constant interest rate.	Financial
"Rate" on page 37	Returns the compound interest rate per period required for an investment to grow from present to future value in a given period.	Financial
"Ref" on page 44	Returns a reference to an existing object.	Miscellaneous
"Replace" on page 54	Replaces all occurrences of one string with another within a specified string.	String
"Right" on page 54	Extracts a number of characters from a given string, beginning with the last character on the right.	String
"Round" on page 16	Evaluates a given numeric value or expression and returns a number rounded to the given number of decimal places.	Arithmetic
"Rtrim" on page 55	Returns a string with all trailing white space characters removed.	String
"Space" on page 55	Returns a string consisting of a given number of blank spaces.	String

Function	**Description**	**Type**
"Str" on page 56	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" on page 57	Inserts a string into another string.	String
"Substr" on page 57	Extracts a portion of a given string.	String
"Sum" on page 16	Returns the sum of the non-null elements of a given set of numbers.	Arithmetic
"Term" on page 38	Returns the number of periods needed to reach a given future value from periodic constant payments into an interest-bearing account.	Financial
"Time" on page 28	Returns the current system time as the number of milliseconds since the "Epoch" on page 18.	Date and Time
"Time2Num" on page 29	Returns the number of milliseconds since the "Epoch" on page 18, given a time string.	Date and Time
"TimeFmt" on page 29	Returns a time format, given a time format style.	Date and Time
"UnitType" on page 44	Returns the units of a unitspan. A unitspan is a string consisting of a number followed by a unit name.	Miscellaneous
"UnitValue" on page 45	Returns the numeric value of a measurement with its associated unitspan, after an optional unit conversion.	Miscellaneous
"Upper" on page 59	Converts all lowercase characters within a string to uppercase.	String
"Uuid" on page 58	Returns a Universally Unique Identifier (UUID) string to use as an identification method.	String
"Within" on page 41	Returns true (1) if the test value is within a given range, and false (0) if it is not.	Logical
"WordNum" on page 59	Returns the English text equivalent of a given number.	String

Chapter 4: 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 "Number literals" on page 2.*

Examples

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

Expression	Returns
`Abs(1.03)`	`1.03`
`Abs(-1.03)`	`1.03`
`Abs(0)`	`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 "Number literals" on page 2.*

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" on page 2.*

Examples

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

Expression	Returns
`Ceil(2.5875)`	`3`
`Ceil(-5.9)`	`-5`
`Ceil("abc")`	`0`
`Ceil(A)`	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 "Number literals" on page 2.*

Examples

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

Expression	Returns
`Count("Tony", "Blue", 41)`	`3`
`Count(Customers[*])`	The number of non-null occurrences of `Customers`.
`Count(Coverage[2], "Home", "Auto")`	3, provided the third occurrence of `Coverage` 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 "Number literals" on page 2.*

Examples

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

Expression	Returns
`Floor(21.3409873)`	`21`
`Floor(5.999965342)`	`5`
`Floor(3.2 * 15)`	`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 "Number literals" on page 2.*

Examples

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

Expression	**Returns**
`Max(234, 15, 107)`	`234`
`Max("abc", 15, "Tony Blue")`	`15`
`Max("abc")`	`0`
`Max(Field1[*], Field2[0])`	Evaluates the non-null occurrences of `Field1` as well as the first occurrence of `Field2`, and returns the highest value.
`Max(Min(Field1[*], Field2[0]), Field3, Field4)`	The first expression evaluates the non-null occurrences of `Field1` as well as the first occurrence of `Field2`, and returns the lowest value. The final result is the maximum of the returned value compared against the values of `Field3` and `Field4`. See also "Min" on page 14.

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 "Number literals" on page 2.*

Examples

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

Expression	**Returns**
`Min(234, 15, 107)`	`15`
`Min("abc", 15, "Tony Blue")`	`15`
`Min("abc")`	`0`
`Min(Field1[*], Field2[0])`	Evaluates the non-null occurrences of `Sales_July` as well as the first occurrence of`Sales_August`, and returns the lowest value.
`Min(Max(Field1[*], Field2[0]), Field3, Field4)`	The first expression evaluates the non-null occurrences of `Field1` as well as the first occurrence of`Field2`, and returns the highest value. The final result is the minimum of the returned value compared against the values of `Field3` and `Field4`. See also "Max" on page 14.

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" on page 2.*

Examples

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

Expression	**Returns**
`Mod(64, -3)`	`1`
`Mod(-13,3)`	`-1`

Expression	Returns
`Mod("abc", 2)`	`0`
`Mod(X[0], Y[9])`	The first occurrence of `X` is used as the dividend and the tenth occurrence of `Y` is used as the divisor.
`Mod(Round(Value[4], 2), Max(Value[*]))`	The first fifth occurrence of `Value` rounded to two decimal places is used as the dividend and the highest of all non-null occurrences of `Value` is used as the divisor. See also "Max" on page 14 and "Round" on page 16.

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" on page 2.*

Examples

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

Expression	Returns
`Round(12.389764537, 4)`	`12.3898`
`Round(20/3, 2)`	`6.67`
`Round(8.9897, "abc")`	9
`Round(FV(400, 0.10/12, 30*12), 2)`	`904195.17`. This takes the value evaluated using the `FV` function and rounds it to two decimal places. See also "FV" on page 32.
`Round(Total_Price, 2)`	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 "Number literals" on page 2.*

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 of `Amount`.
`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 of `Amount`. See also "Max" on page 14, "Min" on page 14, and "Round" on page 16.

Chapter 5: 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
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

Date symbol	Description	Formatted value for English (USA) locale where the locale-sensitive input value is 1/1/08 (which is January 1, 2008)
`E`	One-digit (1-7) day of the week, where (1=Sunday)	3 (because January 1, 2008 is a Tuesday)
`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)
`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

Time symbol	**Description**	**Locale-sensitive input value**	**Formatted value for English (USA) locale**
`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 ***Note:*** *You must use this symbol with an hour symbol.*	2:08 PM	8
`MM`	Zero-padded, two-digit (00-59) minute of the hour ***Note:*** *You must use this symbol with an hour symbol.*	2:08 PM	08
`S`	One- or two-digit (0-59) second of the minute ***Note:*** *You must use this symbol with an hour and minute symbol.*	2:08:09 PM	9
`SS`	Zero-padded, two-digit (00-59) second of the minute ***Note:*** *You must use this symbol with an hour and minute symbol.*	2:08:09 PM	09
`FFF`	Three- digit (000-999) thousandth of the second ***Note:*** *You must use this symbol with an hour, minute, and seconds symbol.*	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) ***Note:*** *You must use this symbol with an hour symbol.*	2:08:09 PM	-0400
`zz`	Alternative ISO-8601 time-zone format (for example, `Z`, +05:00, -00:30, -01, +01:00) ***Note:*** *You must use this symbol with an hour symbol.*	2:08:09 PM	-04:00
`Z`	Abbreviated time-zone name (for example, GMT, GMT+05:00, GMT-00:30, EST, PDT) ***Note:*** *You must use this symbol with an hour symbol.*	2:08:09 PM	EDT

Reserved symbols

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

Symbol	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.

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
`Date()`	`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
`d`	A date string in the format supplied by `f` that also conforms to the locale given by `k`.
`f` (optional)	A date format string. If `f` is omitted, the default date format `MMM D, YYYY` 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.

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
`Date2Num("Mar 15, 1996")`	`35138`
`Date2Num("1/1/1900", "D/M/YYYY")`	`1`
`Date2Num("03/15/96", "MM/DD/YY")`	`35138`
`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)	An integer identifying the locale-specific time format style as follows: • 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 `DateFmt` function:

Expression	Returns
`DateFmt(1)`	`M/D/YY` (if en_US locale is set)

Expression	Returns
`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
`IsoDate2Num("1900")`	`1`
`IsoDate2Num("1900-01")`	`1`
`IsoDate2Num("1900-01-01")`	`1`
`IsoDate2Num("19960315T20:20:20")`	`35138`
`IsoDate2Num("2000-03-01") - IsoDate2Num("20000201")`	`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**
`IsoTime2Num("00:00:00Z")`	`1`, for a user in the Eastern Time (ET) zone.
`IsoTime2Num("13")`	`64800001`, for a user located in Boston, U.S.
`IsoTime2Num("13:13:13")`	`76393001`, for a user located in California.
`IsoTime2Num("19111111T131313+01")`	`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: • 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**
`LocalDateFmt(1, "de_DE")`	`tt.MM.uu`
`LocalDateFmt(2, "fr_CA")`	`aa-MM-jj`
`LocalDateFmt(3, "de_CH")`	`t. MMMM jjjj`
`LocalDateFmt(4, "fr_FR")`	`EEEE j MMMM aaaa`

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: • 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
`LocalTimeFmt(1, "de_DE")`	`HH:mm`
`LocalTimeFmt(2, "fr_CA")`	`HH:mm:ss`
`LocalTimeFmt(3, "de_CH")`	`HH:mm:ss z`
`LocalTimeFmt(4, "fr_FR")`	`HH' h 'mm z`

Num2Date

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

Syntax

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

Parameters

Parameter	Description
`n`	An integer representing the number of days. If `n` is invalid, the function returns an error.
`f` (Optional)	A date format string. If you do not include a value for `f`, the function uses the default date format `MMM D, YYYY`.
`k` (Optional)	A locale identifier string that conforms to the locale naming standards. If you do not include a value for `k`, or if `k` 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
`Num2Date(1, "DD/MM/YYYY")`	`01/01/1900`
`Num2Date(35139, "DD-MMM-YYYY", "de_DE")`	`16-Mrz-1996`
`Num2Date(Date2Num("Mar 15, 2000") - Date2Num("98-03-15", "YY-MM-DD", "fr_CA"))`	`Jan 1, 1902`

Num2GMTime

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

Syntax

`Num2GMTime(`*n* `[,`*f* `[,` *k* `]])`

Parameters

Parameter	Description
`n`	An integer representing the number of milliseconds. If `n` is invalid, the function returns an error.
`f` (Optional)	A time format string. If you do not include a value for `f`, the function uses the default time format `H:MM:SS A`.
`k` (Optional)	A locale identifier string that conforms to the locale naming standards. If you do not include a value for `k`, or if `k` 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 `Num2GMTime` function:

Expression	Returns
`Num2GMTime(1, "HH:MM:SS")`	`00:00:00`
`Num2GMTime(65593001, "HH:MM:SS Z")`	`18:13:13 GMT`
`Num2GMTime(43993001, TimeFmt(4, "de_DE"), "de_DE")`	`12.13 Uhr GMT`

Num2Time

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

Syntax

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

Parameters

Parameter	Description
`n`	An integer representing the number of milliseconds. If `n` is invalid, the function returns an error.
`f` (Optional)	A time format string. If you do not include a value for `f`, the function uses the default time format `H:MM:SS A`.
`k` (Optional)	A locale identifier string that conforms to the locale naming standards. If you do not include a value for `k`, or if `k` 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
`Num2Time(1, "HH:MM:SS")`	`00:00:00` in Greenwich, England and `09:00:00` in Tokyo.
`Num2Time(65593001, "HH:MM:SS Z")`	`13:13:13` EST in Boston, U.S.
`Num2Time(65593001, "HH:MM:SS Z", "de_DE")`	`13:13:13 GMT-05:00` to a German-Swiss user in Boston, U.S.
`Num2Time(43993001, TimeFmt(4, "de_DE"), "de_DE")`	`13.13 Uhr GMT+01:00` to a user in Zurich, Austria.
`Num2Time(43993001, "HH:MM:SSzz")`	`13:13+01:00` to 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
`Time()`	`71533235` at 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
`d`	A time string in the format supplied by `f` that also conforms to the locale given by `k`.
`f` (Optional)	A time format string. If you do not include a value for `f`, the function uses the default time format `H:MM:SS A`.
`k` (Optional)	A locale identifier string that conforms to the locale naming standards. If you do not include a value for `k`, or if `k` 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
`Time2Num("00:00:00 GMT", "HH:MM:SS Z")`	`1`
`Time2Num("1:13:13 PM")`	`76393001` to a user in California on Pacific Standard Time, and `76033001` when that same user is on Pacific Daylight Savings Time.
`Time2Num("13:13:13", "HH:MM:SS") - Time2Num("13:13:13 GMT", "HH:MM:SS Z")) / (60 * 60 * 1000)`	`8` to a user in Vancouver and `5` to a user in Ottawa when on Standard Time. On Daylight Savings Time, the returned values are `7` and `4`, respectively.
`Time2Num("13:13:13 GMT", "HH:MM:SS Z", "fr_FR")`	`47593001`

TimeFmt

Returns a time format, given a time format style.

Syntax

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

Parameters

Parameter	Description
`n` (Optional)	An integer identifying the locale-specific time format style as follows: • 1 (Short style) • 2 (Medium style) • 3 (Long style) • 4 (Full style) If you do not include a value for `n`, or if `n` is invalid, the function uses the default style value.
`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 `TimeFmt` function:

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

Chapter 6: 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 "Number literals" on page 2.*

Examples

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

Expression	Returns
`Apr(35000, 269.50, 360)`	`0.08515404566` for 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 "Number literals" on page 2.*

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.
`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 "Number literals" on page 2.*

Examples

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

Expression	**Returns**
`FV(400, 0.10 / 12, 30 * 12)`	`904195.16991842445`. This is the value, after 30 years, of a $400 a month investment growing at 10% annually.
`FV(1000, 0.075 / 4, 10 * 4)`	`58791.96145535981`. This is the value, after 10 years, of a $1000 a month investment growing at 7.5% a quarter.
`FV(Payment[0], Int_Rate / 4, Time)`	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 "Number literals" on page 2.*

Examples

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

Expression	Returns
`IPmt(30000, 0.085, 295.50, 7, 3)`	`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.
`IPmt(160000, 0.0475, 980, 24, 12)`	`7103.80833569485`. The amount of interest repaid during the third year of the loan.
`IPmt(15000, 0.065, 65.50, 15, 1)`	`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" on page 2.*

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 "Number literals" on page 2.*

Examples

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

Expression	Returns
`Pmt(150000, 0.0475 / 12, 25 * 12)`	`855.17604207164`, which is the monthly payment on a $150,000 loan at 4.75% annual interest, repayable over 25 years.
`Pmt(25000, 0.085, 12)`	`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.
`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 "Number literals" on page 2.*

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 "Number literals" on page 2.*

Examples

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

Expression	Returns
`PV(400, 0.10 / 12, 30 * 12)`	`45580.32799074439`. This is the value after 30 years, of a $400 a month investment growing at 10% annually.
`PV(1000, 0.075 / 4, 10 * 4)`	`58791.96145535981`. This is the value after ten years of a $1000 a month investment growing at 7.5% a quarter.
`PV(Payment[0], Int_Rate / 4, Time)`	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 "Number literals" on page 2.*

Examples

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

Expression	**Returns**
`Rate(12000, 8000, 5)`	`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.
`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 "Number literals" on page 2.*

Examples

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

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

Chapter 7: 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
`n`	The position of the value you want to select within the set. If this value is not a whole number, the function rounds `n` down to the nearest whole value. The function returns an empty string if either of the following conditions is true: • `n` is less than 1. • `n` is greater than the number of items in the set. If `n` is null, the function returns null.
`s1`	The first value in the set of values.
`s2` (Optional)	Additional values in the set.

Examples

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

Expression	Returns
`Choose(3, "Taxes", "Price", "Person", "Teller")`	`Person`
`Choose(2, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)`	`9`
`Choose(Item_Num[0], Items[*])`	Returns the value within the set `Items` that corresponds to the position defined by the first occurrence of `Item_Num`.
`Choose(20/3, "A", "B", "C", "D", "E", "F", "G", "H")`	`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
`Exists(Item)`	True (`1`) if the object `Item` exists and false (`0`) otherwise.
`Exists("hello world")`	False (`0`). The string is not a reference syntax.
`Exists(Invoice.Border.Edge[1].Color)`	True (`1`) if the object `Invoice` exists and has a `Border` property, which in turn has at least one `Edge` property, which in turn has a `Color` 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
`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 `HasValue` function.

Expression	Returns
`HasValue(2)`	True (`1`)
`HasValue(" ")`	False (`0`)
`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`)
`Oneof(3, 1, 25)`	False(`0`)
`Oneof("loan", Fields[*])`	Verifies whether any occurrence of `Fields` has a value of `loan`.

Within

Determines whether the given value is within a given range.

Syntax

`Within(s1, s2, s3)`

Parameters

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

Examples

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

Expression	**Returns**
`Within("C", "A", "D")`	True (`1`)
`Within(1.5, 0, 2)`	True (`1`)
`Within(-1, 0, 2)`	False (`0`)
`Within($, 1, 10)`	True (`1`) if the current value is between 1 and 10.

Chapter 8: 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`	A valid string representing an expression or list of expressions.
	The `Eval` function cannot refer to user-defined variables and functions. For example:
	`var s = "var t = concat(s, ""hello"")"`
	`eval(s)`
	In this case, the `Eval` function does not recognize `s`, and so returns an error. Any subsequent functions that make reference to the variable `s` also fail.

Examples

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

Expression	Returns
`eval("10*3+5*4")`	`50`
`eval("hello")`	`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**
`Null()`	`null`
`Null() + 5`	`5`
`Quantity = Null()`	Assigns `null` to the object `Quantity`.
`Concat("ABC", Null(), "DEF")`	`ABCDEF` See also "Concat" on page 47.

Ref

Returns a reference to an existing object.

Definition

Ref(*v*)

Parameters

Parameters	**Description**
`v`	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**
`Ref("10*3+5*4")`	`10*3+5*4`
`Ref("hello")`	`hello`

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`	A valid string containing a numeric value and a valid unit of measurement (unitspan). Recognized units of measurement are: • in, inches • mm, millimeters • cm, centimeters • pt, points • pc, picas • mp, millipoints If `s` is invalid, the function returns `in`.

Examples

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

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

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

Parameters	**Description**
`s1`	A valid string containing a numeric value and a valid unit of measurement (unitspan). Recognized units of measurement are: • 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`
`UnitValue("6", "pt")`	`432`
`UnitValue("A", "cm")`	`0`
`UnitValue(Size[2], "mp")`	Returns the measurement value of the third occurrence of `Size` converted into millipoints.
`UnitValue("5.08cm", "kilograms")`	`2`

Chapter 9: 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
`s1`	The source string.
`s2`	The search string. If `s2` is not a part of `s1`, the function returns `0`. If `s2` is empty, the function returns `1`.

Examples

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

Expression	Returns
`At("ABCDEFGH", "AB")`	`1`
`At("ABCDEFGH", "F")`	`6`
`At(23412931298471, 29)`	`5`, the first occurrence of `29` within the source string.
`At(Ltrim(Cust_Info[0]), "555")`	The location of the string `555` within the first occurrence of `Cust_Info`. See also "Ltrim" on page 52.

Concat

Returns the concatenation of two or more strings.

Syntax

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

Parameters

Parameter	Description
`s1`	The first string in the set.
`s2` (Optional)	Additional strings to append to the set.

Examples

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

Expression	Returns
`Concat("ABC", "DEF")`	`ABCDEF`
`Concat("Tony", Space(1), "Blue")`	`Tony Blue` See also "Space" on page 55.
`Concat("You owe ", WordNum(1154.67, 2), ".")`	`You owe One Thousand One Hundred Fifty-four Dollars And Sixty-seven Cents.` See also "WordNum" on page 59.

Decode

Returns the decoded version of a given string.

Syntax

`Decode(s1 [, s2 ])`

Parameters

Parameter	Description
`s1`	The string to decode.
`s2` (Optional)	A string identifying the type of decoding to perform. The following strings are valid decoding strings: • url (URL decoding) • html (HTML decoding) • xml (XML decoding) If you do not include a value for `s2`, the function uses URL decoding.

Examples

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

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

Encode

Returns the encoded version of a given string.

Syntax

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

Parameters

Parameter	Description
`s1`	The string to encode.
`s2` (Optional)	A string identifying the type of encoding to perform. The following strings are valid encoding strings: • url (URL encoding) • html (HTML encoding) • xml (XML encoding) If you do not include a value for `s2`, the function uses URL encoding.

Examples

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

Expression	Returns
`Encode("""hello, world!""", "url")`	`%22hello,%20world!%22`
`Encode("ÁÂÃÄÅÆ", "html")`	`&#xc1;&#Xc2;&#Xc3;&#xc4;&#xc5;&#xc6;`

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" on page 22.
`s2`	The source data to format. 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: • 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]]]] 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: • 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]]]] For date-time picture formats, the source data must be an ISO date-time string. For numeric picture formats, the source data must be numeric. For text picture formats, the source data must be textual. For compound picture formats, the number of source data arguments must match the number of subelements in the picture.
`s3` (Optional)	Additional source data to format.

Examples

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

Expression	Returns
`Format("MMM D, YYYY", "20020901")`	`Sep 1, 2002`
`Format("$9,999,999.99", 1234567.89)`	`$1,234,567.89` in the U.S. and `1 234 567,89` Euros 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
`s`	The string to extract from.
`n`	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
`Left("ABCDEFGH", 3)`	`ABC`
`Left("Tony Blue", 5)`	`"Tony "`
`Left(Telephone[0], 3)`	The first three characters of the first occurrence of `Telephone`.
`Left(Rtrim(Last_Name), 3)`	The first three characters of `Last_Name`. See also "Rtrim" on page 55.

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
`Len("ABDCEFGH")`	8
`Len(4)`	1
`Len(Str(4.532, 6, 4))`	6 See also “Str” on page 56.
`Len(Amount[*])`	The number of characters in the first occurrence of `Amount`.

Lower

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

Syntax

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

Parameters

Parameter	Description
`s`	The string to convert.
`k` (Optional)	A string representing a valid locale. If you do not include a value for `k`, the function uses the ambient locale. See “Locales” on page 22. 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)

Examples

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

Expression	Returns
`Lower("ABC")`	`abc`
`Lower("21 Main St.")`	`21 main st.`
`Lower(15)`	`15`
`Lower(Address[0])`	This example converts the first occurrence of `Address` 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
`s`	The string to trim.

Examples

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

Expression	Returns
`Ltrim("          ABCD")`	`"ABCD"`
`Ltrim(Rtrim("          Tony Blue          "))`	`"Tony Blue"` See also “Rtrim” on page 55.
`Ltrim(Address[0])`	Removes any leading white space from the first occurrence of `Address`.

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 “Date and Time Functions” on page 18.
`s2`	The string data to parse.

Examples

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

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

Replace

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

Syntax

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

Parameters

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

Examples

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

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

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**
`Right("ABCDEFGH", 3)`	`FGH`
`Right("Tony Blue", 5)`	`" Blue"`
`Right(Telephone[0], 7)`	The last seven characters of the first occurrence of `Telephone`.
`Right(Rtrim(CreditCard_Num), 4)`	The last four characters of `CreditCard_Num`. See also "Rtrim" on page 55.

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**
`s`	The string to trim.

Examples

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

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

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
`Space(5)`	`"     "`
`Space(Max(Amount[*]))`	A blank string with as many characters as the value of the largest occurrence of `Amount`. See also "Max" on page 14.
`Concat("Tony", Space(1), "Blue")`	`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.
`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
`Str(2.456)`	`"         2"`
`Str(4.532, 6, 4)`	`4.5320`
`Str(234.458, 4)`	`" 234"`
`Str(31.2345, 4, 2)`	`****`
`Str(Max(Amount[*]), 6, 2)`	Converts the largest occurrence of `Amount` to a six-character string with two decimal places. See also "Max" on page 14.

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
`Stuff("TonyBlue", 5, 0, " ")`	`Tony Blue`
`Stuff("ABCDEFGH", 4, 2)`	`ABCFGH`
`Stuff(Address[0], Len(Address[0]), 0, "Street")`	This adds the word `Street` onto the end of the first occurrence of `Address`. See also "Len" on page 51.
`Stuff("members-list@myweb.com", 0, 0, "cc:"`	`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
`Substr("ABCDEFG", 3, 4)`	`CDEF`
`Substr(3214, 2, 1)`	`2`
`Substr(Last_Name[0], 1, 3)`	Returns the first three characters from the first occurrence of `Last_Name`.
`Substr("ABCDEFG", 5, 0)`	`""`
`Substr("21 Waterloo St.", 4, 5)`	`Water`

Uuid

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

Syntax

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

Parameters

Parameter	Description
`n`	A number identifying the format of the UUID string. Valid numbers are: • 0 (default value): UUID string only contains hex octets. • 1: UUID string contains dash characters separating the sequences of hex octets at fixed positions. If you do not include a value for `n`, the function uses the default value.

Examples

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

Expression	**Returns**
`Uuid()`	A value such as `3c3400001037be8996c400a0c9c86dd5`
`Uuid(0)`	A value such as `3c3400001037be8996c400a0c9c86dd5`
`Uuid(1)`	A value such as `1a3ac000-3dde-f352-96c4-00a0c9c86dd5`
`Uuid(7)`	A value such as `1a3ac000-3dde-f352-96c4-00a0c9c86dd5`

Upper

Converts all lowercase characters within a string to uppercase.

Syntax

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

Parameters

Parameter	**Description**
`s`	The string to convert. See “Locales” on page 22.
`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**
`Upper("abc")`	`ABC`
`Upper("21 Main St.")`	`21 MAIN ST.`
`Upper(15)`	`15`
`Upper(Address[0])`	This example converts the first occurrence of `Address` to all uppercase letters.

WordNum

Returns the English text equivalent of a given number.

Syntax

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

Parameters

Parameter	Description
`n1`	The number to convert. If any of the following statements is true, the function returns * (asterisk) characters to indicate an error: • `n1` is not a number. • The integral value of `n1` is negative. • The integral value of `n1` is greater than 922,337,203,685,477,550.
`n2` (Optional)	A number identifying the formatting option. Valid numbers are: • 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 `n2`, the function uses the default value (`0`).
`k` (Optional)	A string representing a valid locale. If you do not include a value for `k`, the function uses the ambient locale. See “Locales” on page 22. 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
`WordNum(123.45)`	`One Hundred and Twenty-three Dollars`
`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.

Chapter 10: 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
`Get("http://www.myweb.com/data/mydata.xml")`	XML data taken from the specified file.
`Get("ftp://ftp.gnu.org/gnu/GPL")`	The contents of the GNU Public License.
`Get("http://intranet?sql=SELECT+*+FROM+projects+FOR+XML+AUTO,+ELEMENTS")`	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.
`s3` (Optional)	A string containing the content type of the data to post. Here are valid content types: • application/octet-stream (default value) • text/html • text/xml • text/plain • multipart/form-data • application/x-www-form-urlencoded • Any other valid MIME type If you do not include a value for `s3`, 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.
`s4` (Optional)	A string containing the name of the code page used to encode the data. Here are valid code page names: • UTF-8 (default value) • UTF-16 • ISO-8859-1 • Any character encoding listed by the Internet Assigned Numbers Authority (IANA) If you do not include a value for `s4`, 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.
`s5` (Optional)	A string containing any additional HTTP headers to be included with the posting of the data. If you do not include a value for `s5`, the function does not include an additional HTTP header in the post. SOAP servers usually require a SOAPAction header when posting to them.

Examples

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

Expression	Returns
`Post("http://tools_build/scripts/jfecho.cgi", "user=joe&passwd=xxxxx&date=27/08/2002", "application/x-www-form-urlencoded")`	Posts some URL encoded login data to a server and returns that server's acknowledgement page.
`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")`	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.
`s3` (Optional)	A string containing the name of the code page used to encode the data. Here are valid code page names: • UTF-8 (default value) • UTF-16 • ISO8859-1 • Any character encoding listed by the Internet Assigned Numbers Authority (IANA) 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.

Examples

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

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

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.*