



# FormCalc Reference

**Adobe Designer**  
Version 6.0

© 2004 Adobe Systems Incorporated. All rights reserved.

Adobe® Designer Documentation for Microsoft® Windows®  
SAP Version May 2004

As of April 12, 2002, Accelio Corporation (formerly JetForm Corporation) was purchased by Adobe Systems Incorporated. As of that date, any reference to JetForm or Accelio shall be deemed to refer to Adobe Systems Incorporated.

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Acrobat, and Acrobat Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Microsoft and Windows are either trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks are the property of their respective owners.

This software is based in part on the work of the Independent JPEG group. Portions © 1995-1996 Access Softtek Inc. All rights reserved.

This software is based in part on the work of the FreeType team.

This product includes code licensed from RSA Security, Inc. Some portions licensed from IBM are available at <http://oss.software.IBM.com/icu4s/>.

Software included in this program may contain an implementation of the LZW algorithm licensed under the foreign counterparts to expired U.S. Patent 4,558,302.

The Proximity/Merriam-Webster, Inc. Linguibase. Copyright 1983, 1990 Merriam Webster, Inc. Copyright 1983, 1990. All rights reserved. Proximity Technology.

Portions copyright 1992-1995 Summit Software Company.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

---

# Contents

---

<b>Preface .....</b>	<b>6</b>
What's in this Guide? .....	6
Who should read this guide? .....	6
Related documentation .....	6
<b>1 Overview .....</b>	<b>7</b>
About scripting in Adobe Designer .....	7
<b>2 Language Reference .....</b>	<b>8</b>
Building blocks .....	8
Literals .....	8
Number literals .....	8
String literals .....	9
Operators .....	10
Comments .....	11
Keywords .....	11
Identifiers .....	12
Variables .....	12
Line terminators .....	13
White space .....	13
Expressions .....	13
Simple .....	14
Promoting operands .....	15
Assignment .....	16
Logical OR .....	16
Logical AND .....	16
Unary .....	17
Equality and inequality .....	17
Relational .....	18
If expressions .....	19
Accessors .....	20
Method calls .....	24
Function calls .....	25
<b>3 Alphabetical Functions List .....</b>	<b>26</b>
<b>4 Arithmetic Functions .....</b>	<b>30</b>
Abs .....	31
Avg .....	32
Ceil .....	33
Count .....	34
Floor .....	35
Max .....	36
Min .....	37
Mod .....	38
Round .....	39
Sum .....	40

<b>5</b>	<b>Date and Time Functions .....</b>	<b>41</b>
	Structuring dates and times .....	42
	Locales .....	42
	Epoch.....	43
	Date formats .....	43
	Time formats.....	44
	Date and time picture formats .....	44
	Date .....	47
	Date2Num.....	48
	DateFmt .....	49
	IsoDate2Num .....	50
	IsoTime2Num.....	51
	LocalDateFmt.....	52
	LocalTimeFmt .....	53
	Num2Date.....	54
	Num2GMTTime .....	55
	Num2Time .....	56
	Time.....	57
	Time2Num .....	58
	TimeFmt.....	59
<b>6</b>	<b>Financial Functions.....</b>	<b>60</b>
	Apr .....	61
	CTerm .....	62
	FV.....	63
	IPmt .....	64
	NPV .....	65
	Pmt .....	66
	PPmt.....	67
	PV .....	68
	Rate.....	69
	Term .....	70
<b>7</b>	<b>Logical Functions.....</b>	<b>72</b>
	Choose.....	73
	Exists.....	74
	HasValue.....	75
	Oneof .....	76
	Within .....	77
<b>8</b>	<b>Miscellaneous Functions .....</b>	<b>78</b>
	Eval.....	79
	Null.....	80
	Ref .....	81
	UnitType .....	82
	UnitValue.....	83
<b>9</b>	<b>String Functions.....</b>	<b>84</b>
	At .....	85
	Concat .....	86
	Decode .....	87
	Encode.....	88

**9 String Functions (Continued)**

Format .....	89
Left .....	90
Len .....	91
Lower .....	92
Ltrim .....	93
Parse .....	94
Replace .....	95
Right .....	96
Rtrim .....	97
Space .....	98
Str .....	99
Stuff .....	100
Substr .....	101
Uuid .....	102
Upper .....	103
WordNum .....	104

**10 URL Functions ..... 106**

Get .....	107
Post .....	108
Put .....	110

**Index ..... 111**

---

## Preface

---

Adobe® Designer 6.0 provides a set of tools that enables a form developer to build intelligent business documents. The form developer can, optionally, incorporate scripting to create a richer experience for the recipient of the form. For example, you might use simple calculations to automatically update costs on a purchase order, or you might use more advanced scripting to modify the appearance of your form in response to the locale of the user.

Designer supports both FormCalc, a calculation language created by Adobe, and JavaScript, a powerful and popular scripting language.

### What's in this Guide?

This guide acts as a reference for the FormCalc language, and contains detailed material on all FormCalc functions, which are organized into chapters according to function category. This guide also includes an alphabetical listing of all functions with a brief description of their purpose for quick referencing.

### Who should read this guide?

This guide provides information to assist form developers interested in using the FormCalc language to create calculations to enhance their form designs. FormCalc is simple and accessible for those with little or no scripting experience. It also follows many rules and conventions common to other scripting languages, so experienced form developers will find their skills relevant to using FormCalc.

### Related documentation

For additional information on using the FormCalc calculation language, see the section Using FormCalc in the Adobe Designer Help.

FormCalc is a simple yet powerful calculation language modeled on common spreadsheet software. Its purpose is to facilitate fast and efficient form design without requiring a knowledge of traditional scripting techniques or languages. Users new to FormCalc can expect, with the use of a few built-in functions, to quickly create forms that save end users from performing time-consuming calculations, validations, and other verifications. In this way, a form author is able to create a basic intelligence around a form at design time that allows the resulting interactive form to react according to the data it encounters.

The built-in functions that make up 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 purpose of the functions is to provide quick and easy manipulation of the data in a useful way.

## About scripting in Adobe Designer

Within Designer, FormCalc is the default scripting language in all scripting locations, with JavaScript as the alternative. Scripting takes place on the various events that accompany each form object, and it is possible to use a mixture of FormCalc and JavaScript on interactive, static, and dynamic forms. However, if you are using a server-based process (for example Adobe® document services to create forms for viewing in an internet browser, FormCalc scripts on certain form object events do not render onto the HTML form. This is to prevent internet browser errors from occurring when users work with the completed form.

## Building blocks

The smallest components of the FormCalc language provide the starting point for writing expressions. Each FormCalc expression is essentially a sequence of some combination of these components.

- ["Literals" on page 8](#)
- ["Operators" on page 10](#)
- ["Comments" on page 11](#)
- ["Keywords" on page 11](#)
- ["Identifiers" on page 12](#)
- ["Variables" on page 12](#)
- ["Line terminators" on page 13](#)
- ["White space" on page 13](#)

## 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: 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 use two quotation marks instead of one for both the leading and trailing quotation marks. For example:

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

All Unicode characters have an equivalent 6 character string 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 operators are symbols common to most other scripting languages, including unary, multiplicative, additive, relational, equality, logical operators, 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 runtime.

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"
```

## Keywords

FormCalc restricts the use of certain words and abbreviations. These keywords are used as accessors, parts of expressions, special number literals, and operators.

The following table lists the entire set of 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	

**Note:** These keywords are case-insensitive. FormCalc reserves all possible forms of these words.

## Identifiers

An identifier is a sequence of characters of unlimited length that denotes either a function or a method name. Each identifier must begin 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. Case-sensitivity is mandated by the FormCalc hosting environments.

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

The following are examples of valid identifiers:

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

## 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 defines the `userName` variable and sets 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 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 supercede a similarly named variable you define in the Variables tab of the Form Properties dialog.

## Line terminators

Line terminators are primarily for separating sections of output and improving readability.

The following table lists the entire set of 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 generally irrelevant during FormCalc processing.

Character name	Unicode character
Form Feed	#xC
Horizontal Tab	#x9
Space	#x20
Vertical Tab	#xB

## 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`.

**Note:** There are no characters used to delimit the end of expressions. Each new expression within a particular calculation should occupy its own separate line.

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

- ["Simple" on page 14](#)
- ["Assignment" on page 16](#)
- ["Logical OR" on page 16](#)
- ["Logical AND" on page 16](#)
- ["Unary" on page 17](#)
- ["Equality and inequality" on page 17](#)
- ["Relational" on page 18](#)
- ["If expressions" on page 19](#)

## Simple

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

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

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
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 priority when they appear within expressions. The following table illustrates this operator hierarchy:

Priority	Operator
Highest	=
	(Unary) -, +, not
	*, /
	+, -
	<, <=, >, >=, lt, le, gt, ge

Priority	Operator
	<code>==</code> , <code>&lt;&gt;</code> , <code>eq</code> , <code>ne</code>
	<code>&amp;</code> , <code>and</code>
Lowest	<code> </code> , <code>or</code>

## 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
<code>(5 - "abc") * 3</code>	<code>(5 - 0) * 3</code>	15
<code>"100" / 10e1</code>	<code>100 / 10e1</code>	1
<code>5 + null + 3</code>	<code>5 + 0 + 3</code>	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 accessor 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 accessors, see ["Accessors" on page 20](#).

## Logical OR

A logical OR expression returns either true (1) if at least one of the 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

The following 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 the operands is false (0). If both operands are null, the expression returns null.

Expression	Character representation
Logical AND	& and

The following 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, “not nothing” should be something.

The following are examples of using the unary expression:

Expression	Returns
- (17)	-17
- (-17)	17
+ (17)	17
+ (-17)	-17
not ("true")	1 (true)
not (1)	0 (false)

## Equality and inequality

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

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

The following are examples of using the equality and inequality expressions:

Expression	Returns
3 == 3	1 (true)
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 comparison of the 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.

The following 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><b>Note:</b> 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>

The following are examples of using the if expression:

Expression	Returns
<pre>if ( 1 &lt; 2 ) then   1 endif</pre>	1
<pre>if ( "abc" &gt; "def") then   1 and 0 else   0 endif</pre>	0
<pre>if ( Field1 &lt; Field2 ) then   Field3 = 0 elseif ( Field1 &gt; 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.

## Accessors

FormCalc provides access to form design object properties and values. An accessor is a mechanism that enables you to assign or retrieve specific object values and properties.

The following example demonstrates both assigning and retrieving object values:

```
Invoice.VAT = Invoice.Total * (8 / 100)
```

In this case the accessor `Invoice.VAT` is assigned the value of `Invoice.Total * (8 / 100)`.

In the context of form design, a qualified hierarchy enables access to all the objects on the form design. Accessors provide predefined syntax that makes navigation of this hierarchy easier.

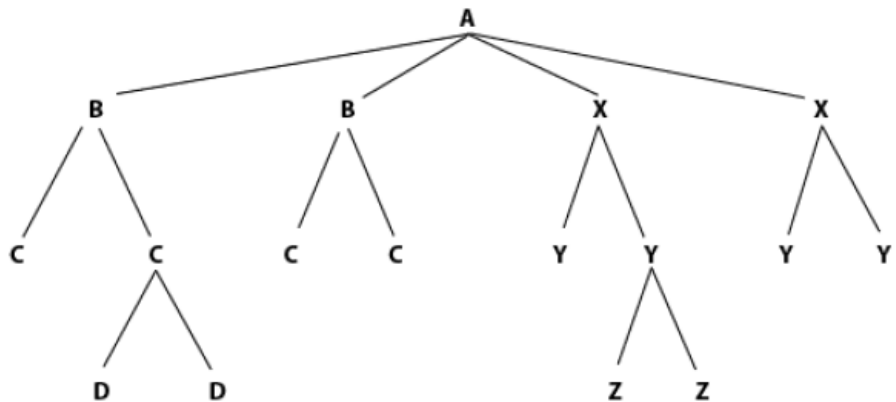
The following table outlines the correct syntax for all accessors within FormCalc.

Notation	Description
\$	<p>Refers to the current node. For example:</p> <pre>\$ .A</pre> <p>The above example finds the node <code>A</code> as an immediate child of the current node. Note that this is different from the following example:</p> <pre>A</pre> <p>In this second example, if node <code>A</code> is not an immediate child of the current node, but if it does not exist there, the tree is searched.</p> <p><b>Note:</b> This accessor must appear at the beginning of a hierarchy reference, that is, before the first period.</p>

Notation	Description
!	<p>Represents the first two nodes of the data model, xfa.datasets. For example:</p> <pre>!dbresults</pre> <p>is equivalent to:</p> <pre>xfa.datasets.dbresults</pre> <p><b>Note:</b> This accessor must appear at the beginning of a hierarchy reference, that is, before the first period.</p>
\$data	<p>Represents the root node of the data model, xfa.datasets.data. For example:</p> <pre>\$data.purchase_order.total</pre> <p>is equivalent to:</p> <pre>xfa.datasets.data.purchase_order.total</pre> <p><b>Note:</b> This accessor must appear at the beginning of a hierarchy reference, that is, before the first period.</p>
\$template	<p>Represents the root node of the template model, xfa.template. For example:</p> <pre>\$template.purchase_order.item[1]</pre> <p>is equivalent to:</p> <pre>xfa.template.purchase_order.item[1]</pre> <p><b>Note:</b> This accessor must appear at the beginning of a hierarchy reference, that is, before the first period.</p>
\$form	<p>Represents the root node of the form model, xfa.form. For example:</p> <pre>\$form.purchase_order.tax[0]</pre> <p>is equivalent to:</p> <pre>xfa.form.purchase_order.tax[0]</pre> <p><b>Note:</b> This accessor must appear at the beginning of a hierarchy reference, that is, before the first period.</p>
\$record	<p>Represents the current record of a collection of data, such as from an XML file. For example:</p> <pre>\$record.header.txtOrderedByCity</pre> <p>references the <code>txtOrderedByCity</code> node within the <code>header</code> node of the current XML data.</p> <p><b>Note:</b> This accessor must appear at the beginning of a hierarchy reference, that is, before the first period.</p>
\$event	<p>Represents the current form object event. For example:</p> <pre>\$event.name</pre> <p>is equivalent to:</p> <pre>xfa.event.name</pre> <p><b>Note:</b> This accessor must appear at the beginning of a hierarchy reference, that is, before the first period.</p>

Notation	Description
*	<p>Selects all child nodes regardless of name. It cannot begin an XFA-SOM expression. For example, this expression selects all containers on a form design:</p> <pre>\$template.*</pre>
..	<p>At any point after the first period, an ellipsis (..) can be used to search for descendant nodes (at any depth) of a particular name. For example, the expression <code>A..B</code> means locate the node <code>A</code>, and find a descendant of <code>A</code> called <code>B</code>.</p> <pre> graph TD     A --&gt; B1[B]     A --&gt; B2[B]     A --&gt; X1[X]     A --&gt; X2[X]     B1 --&gt; C1[C]     B1 --&gt; C2[C]     B2 --&gt; C3[C]     B2 --&gt; C4[C]     X1 --&gt; Y1[Y]     X1 --&gt; Y2[Y]     X2 --&gt; Y3[Y]     X2 --&gt; Y4[Y]     Y1 --&gt; Z1[Z]     Y1 --&gt; Z2[Z]     Y2 --&gt; Z3[Z]     Y2 --&gt; Z4[Z] </pre> <p>Using the example tree above:</p> <pre>A..C</pre> <p>is equivalent to:</p> <pre>A.B[0].C[0]</pre> <p>because <code>C[0]</code> is the first <code>C</code> node FormCalc encounters on its search. As a second example:</p> <pre>A..C[*]</pre> <p>returns the two leftmost <code>C</code> nodes, because in this example, <code>A..C[*]</code> is equivalent to <code>A.B[0].C[*]</code>.</p>
#	<p>Matches an attribute or property. This accessor is useful if both a property and a node, or an attribute and a node, have the same name. The number sign (#) ensures that the property or attribute is accessed. For example:</p> <pre>\$form.purchase_order.tax.#name</pre> <p>This expression returns the actual name of the reference accessor, in this case <code>tax</code>.</p>

Notation	Description
[ ]	<p>An array referencing syntax. FormCalc treats the collection of accessible objects with the same name as an array. Note that all array references are 0-based.</p> <p>A valid array element reference takes the form of one of the following:</p> <ul style="list-style-type: none"><li>• [ <i>n</i> ]</li></ul> <p>Where <i>n</i> is an absolute occurrence index number beginning at 0. An occurrence number that is out of range is an error. Occurrence numbers in SOM syntax are not expressions. Only numbers are valid.</p> <p>For example:</p> <pre>\$xfa.template.Quantity[3]</pre> <p>refers to the fourth occurrence of Quantity.</p> <ul style="list-style-type: none"><li>• [ +/- <i>n</i> ]</li></ul> <p>Where <i>n</i> indicates an occurrence relative to the occurrence of the object making the reference. Positive values yield higher occurrence numbers, while negative values yield lower occurrence numbers. For example:</p> <pre>\$xfa.template.Quantity[+2]</pre> <p>This expression yields the occurrence of <code>Quantity</code> whose occurrence number is two more than the occurrence number of the container making the reference. For example, if this reference was attached to <code>Amount [2]</code>, the reference would be the same as:</p> <pre>\$xfa.template.Quantity[4]</pre> <p>If the computed index number is out of range, it is an error.</p> <p>The most common use of this syntax is for locating the previous or next occurrence of a particular field. For example, every occurrence of the field <code>Amount</code> (except the first) might use <code>Amount [-1]</code> to get at the value of the previous amount field.</p> <ul style="list-style-type: none"><li>• [ * ]</li></ul> <p>Indicates multiple occurrences of the object. The first named object is found, and objects of the same name that are siblings to the first are returned. Note using this notation results in a collection of objects. For example:</p> <pre>Quantity[ * ]</pre> <p>This expression refers to all objects with a name of <code>Quantity</code> that are siblings to the first <code>Quantity</code> found.</p>

Notation	Description
	 <p>Using the tree for reference, these expressions return the following:</p> <ul style="list-style-type: none"> <li> <b>A.B[*]</b>  Both B nodes. </li> <li> <b>A.B.C[*]</b>  The two leftmost C nodes, because A.B resolves to the first B node on the left, and the C[*] is evaluated relative to that node. </li> <li> <b>A.B[*].C</b>  The first and the third C nodes from the left. A.B[*] resolves to both B nodes, and the C is evaluated relative to both of those B nodes. </li> <li> <b>A.B[*].C[1]</b>  The second and fourth C nodes from the left. A.B[*] resolves to both B nodes, and the C[1] is evaluated relative to both of those B nodes. </li> <li> <b>A.B[*].C[*]</b>  All four C nodes. </li> <li> <b>A.*</b>  Both B nodes and both X nodes. </li> <li> <b>A.X.*</b>  The two leftmost Y nodes. </li> </ul>

## Method calls

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

**Note:** Different form design objects support different methods.



## 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 a 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 Functions List” on page 26](#).

### 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
<a href="#">"Abs" on page 31</a>	Returns the absolute value of a numeric value or expression.	Arithmetic
<a href="#">"Apr" on page 61</a>	Returns the annual percentage rate for a loan.	Financial
<a href="#">"At" on page 85</a>	Locates the starting character position of a string within another string.	String
<a href="#">"Avg" on page 32</a>	Evaluates a set of number values and/or expressions and returns the average of the non-null elements contained within that set.	Arithmetic
<a href="#">"CTerm" on page 62</a>	Returns the number of periods needed for an investment earning a fixed, but compounded, interest rate to grow to a future value.	Financial
<a href="#">"Ceil" on page 33</a>	Returns the whole number greater than or equal to a given number.	Arithmetic
<a href="#">"Choose" on page 73</a>	Selects a value from a given set of parameters.	Logical
<a href="#">"Concat" on page 86</a>	Returns the concatenation of two or more strings.	String
<a href="#">"Count" on page 34</a>	Evaluates a set of values and/or expressions and returns the number of non-null elements contained within the set.	Arithmetic
<a href="#">"Date" on page 47</a>	Returns the current system date as the number of days since the <a href="#">epoch</a> .	Date and Time
<a href="#">"Date2Num" on page 48</a>	Returns the number of days since the <a href="#">epoch</a> , given a date string.	Date and Time
<a href="#">"DateFmt" on page 49</a>	Returns a date format string, given a date format style.	Date and Time
<a href="#">"Decode" on page 87</a>	Returns the decoded version of a given string.	String
<a href="#">"Encode" on page 88</a>	Returns the encoded version of a given string.	String
<a href="#">"Exists" on page 74</a>	Determines whether the given parameter is an accessor to an existing object.	Logical
<a href="#">"FV" on page 63</a>	Returns the future value of consistent payment amounts made at regular intervals at a constant interest rate.	Financial
<a href="#">"Floor" on page 35</a>	Returns the largest whole number that is less than or equal to the given value.	Arithmetic

Function	Description	Type
<a href="#">"Format" on page 89</a>	Formats the given data according to the specified picture format string.	String
<a href="#">"Get" on page 107</a>	Downloads the contents of the given URL.	URL
<a href="#">"HasValue" on page 75</a>	Determines whether the given parameter is an accessor with a non-null, non-empty, or non-blank value.	Logical
<a href="#">"IPmt" on page 64</a>	Returns the amount of interest paid on a loan over a set period of time.	Financial
<a href="#">"IsoDate2Num" on page 50</a>	Returns the number of days since the <a href="#">epoch</a> , given an valid date string.	Date and Time
<a href="#">"IsoTime2Num" on page 51</a>	Returns the number of milliseconds since the <a href="#">epoch</a> , given a valid time string.	Date and Time
<a href="#">"Left" on page 90</a>	Extracts a specified number of characters from a string, starting with the first character on the left.	String
<a href="#">"Len" on page 91</a>	Returns the number of characters in a given string.	String
<a href="#">"LocalDateFmt" on page 52</a>	Returns a localized date format string, given a date format style.	Date and Time
<a href="#">"LocalTimeFmt" on page 53</a>	Returns a localized time format string, given a time format style.	Date and Time
<a href="#">"Lower" on page 92</a>	Converts all uppercase characters within a specified string to lowercase characters.	String
<a href="#">"Ltrim" on page 93</a>	Returns a string with all leading white space characters removed.	String
<a href="#">"Max" on page 36</a>	Returns the maximum value of the non-null elements in the given set of numbers.	Arithmetic
<a href="#">"Min" on page 37</a>	Returns the minimum value of the non-null elements of the given set of numbers.	Arithmetic
<a href="#">"Mod" on page 38</a>	Returns the modulus of one number divided by another.	Arithmetic
<a href="#">"NPV" on page 65</a>	Returns the net present value of an investment based on a discount rate and a series of periodic future cash flows.	Financial
<a href="#">"Num2Date" on page 54</a>	Returns a date string, given a number of days since the <a href="#">epoch</a> .	Date and Time
<a href="#">"Num2GMTTime" on page 55</a>	Returns a GMT time string, given a number of milliseconds from the <a href="#">epoch</a> .	Date and Time
<a href="#">"Num2Time" on page 56</a>	Returns a time string, given a number of milliseconds from the <a href="#">epoch</a> .	Date and Time
<a href="#">"Oneof" on page 76</a>	Returns true (1) if a value is in a given set, and false (0) if it is not.	Logical

Function	Description	Type
<a href="#">"Parse" on page 94</a>	Analyzes the given data according to the given picture format.	String
<a href="#">"Pmt" on page 66</a>	Returns the payment for a loan based on constant payments and a constant interest rate.	Financial
<a href="#">"Post" on page 108</a>	Posts the given data to the specified URL.	URL
<a href="#">"PPmt" on page 67</a>	Returns the amount of principal paid on a loan over a period of time.	Financial
<a href="#">"Put" on page 110</a>	Uploads the given data to the specified URL.	URL
<a href="#">"PV" on page 68</a>	Returns the present value of an investment of periodic constant payments at a constant interest rate.	Financial
<a href="#">"Rate" on page 69</a>	Returns the compound interest rate per period required for an investment to grow from present to future value in a given period.	Financial
<a href="#">"Replace" on page 95</a>	Replaces all occurrences of one string with another within a specified string.	String
<a href="#">"Right" on page 96</a>	Extracts a number of characters from a given string, beginning with the last character on the right.	String
<a href="#">"Round" on page 39</a>	Evaluates a given numeric value or expression and returns a number rounded to the given number of decimal places.	Arithmetic
<a href="#">"Rtrim" on page 97</a>	Returns a string with all trailing white space characters removed.	String
<a href="#">"Space" on page 98</a>	Returns a string consisting of a given number of blank spaces.	String
<a href="#">"Str" on page 99</a>	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
<a href="#">"Stuff" on page 100</a>	Inserts a string into another string.	String
<a href="#">"Substr" on page 101</a>	Extracts a portion of a given string.	String
<a href="#">"Sum" on page 40</a>	Returns the sum of the non-null elements of a given set of numbers.	Arithmetic
<a href="#">"Term" on page 70</a>	Returns the number of periods needed to reach a given future value from periodic constant payments into an interest-bearing account.	Financial
<a href="#">"Time" on page 57</a>	Returns the current system time as the number of milliseconds since the <a href="#">epoch</a> .	Date and Time
<a href="#">"Time2Num" on page 58</a>	Returns the number of milliseconds since the <a href="#">epoch</a> , given a time string.	Date and Time

Function	Description	Type
<a href="#">"TimeFmt" on page 59</a>	Returns a time format, given a time format style.	Date and Time
<a href="#">"Uuid" on page 102</a>	Returns a Universally Unique Identifier (UUID) string to use as an identification method.	String
<a href="#">"Upper" on page 103</a>	Converts all lowercase characters within a string to uppercase.	String
<a href="#">"UnitType" on page 82</a>	Returns the units of a unitspan. A unitspan is a string consisting of a number followed by a unit name.	Miscellaneous
<a href="#">"UnitValue" on page 83</a>	Returns the numeric value of a measurement with its associated unitspan, after an optional unit conversion.	Miscellaneous
<a href="#">"Within" on page 77</a>	Returns true (1) if the test value is within a given range, and false (0) if it is not.	Logical
<a href="#">"WordNum" on page 104</a>	Returns the English text equivalent of a given number.	String

# 4

## Arithmetic Functions

---

These functions perform a range of mathematical operations.

### Functions

- ["Abs" on page 31](#)
- ["Avg" on page 32](#)
- ["Ceil" on page 33](#)
- ["Count" on page 34](#)
- ["Floor" on page 35](#)
- ["Max" on page 36](#)
- ["Min" on page 37](#)
- ["Mod" on page 38](#)
- ["Round" on page 39](#)
- ["Sum" on page 40](#)

# Abs

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

**Note:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see ["Number literals" on page 8](#).

## Syntax

Abs ( *n* )

## Parameters

Parameter	Description
<i>n</i>	A numeric value or expression to evaluate.

## Examples

The following are examples of using the Abs function:

Expression	Returns
Abs ( 1 . 03 )	1 . 03
Abs ( -1 . 03 )	1 . 03
Abs ( 0 )	0
Abs ( "abc" )	0
Abs ( Count [ * ] )	The absolute value of the first non-null occurrence of Count.

## Avg

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

**Note:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [“Number literals” on page 8](#).

### Syntax

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

### Parameters

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

### Examples

The following are of using the Avg function:

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



# Ceil

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

**Note:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see ["Number literals" on page 8](#).

## Syntax

`Ceil ( n )`

## Parameters

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

## Examples

The following 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 (Amount [*])</code>	A whole number greater than or equal to the value of the first occurrence of <code>Amount</code> .
<code>Ceil (CTerm(0.10, 500000, 12000))</code>	40 See also <a href="#">"CTerm" on page 62</a> .

## Count

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

**Note:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see ["Number literals" on page 8](#).

### Syntax

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

### Parameters

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

### Examples

The following are examples 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.

**Note:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [“Number literals” on page 8](#).

## Syntax

Floor(*n*)

## Parameters

Parameter	Description
<i>n</i>	Any numeric value or expression.

## Examples

The following are examples of using the Floor function:

Expression	Returns
Floor(21.3409873)	21
Floor(5.999965342)	5
Floor(3.2 * 15)	48
Floor("abc")	0
Floor(Min(-2.65, 12.16, -5.98))	-6 See also <a href="#">“Min” on page 37</a> .

# Max

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

**Note:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [“Number literals” on page 8](#).

## Syntax

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

## Parameters

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

## Examples

The following 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(Sales_July[*], Sales_August[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 highest value.
<code>Max(Min(Sales_July[*], Sales_August[0]), Q2_01, Q3_99)</code>	The first expression 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. The final result is the maximum of this value compared against the values of <code>Q2_01</code> and <code>Q3_99</code> .  See also <a href="#">“Min” on page 37</a> .

# Min

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

**Note:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [“Number literals” on page 8](#).

## Syntax

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

## Parameters

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

## Examples

The following 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(Sales_July[*], Sales_August[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(Sales_July[*], Sales_August[0]), Q2_01, Q3_99)</code>	The first expression evaluates the non-null occurrences of <code>Sales_July</code> as well as the first occurrence of <code>Sales_August</code> , and returns the highest value. The final result is the minimum of this value compared against the values of <code>Q2_01</code> and <code>Q3_99</code> .  See also <a href="#">“Max” on page 36</a> .

# 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.

**Note:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see ["Number literals" on page 8](#).

## Syntax

`Mod ( n1 , n2 )`

## Parameters

Parameter	Description
<i>n1</i>	The dividend, a numeric value or expression.
<i>n2</i>	The divisor, a numeric value or expression.

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

## Examples

The following 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 <i>X</i> is used as the dividend and the tenth occurrence of <i>Y</i> is used as the divisor.
<code>Mod ( Round ( Value [ 4 ] , 2 ) , Max ( Value [ * ] ) )</code>	The first fifth occurrence of <i>Value</i> rounded to two decimal places is used as the dividend and the highest of all non-null occurrences of <i>Value</i> is used as the divisor.  See also <a href="#">"Max" on page 36</a> and <a href="#">"Round" on page 39</a> .

# Round

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

**Note:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see ["Number literals" on page 8](#).

## Syntax

`Round(n1 [, n2])`

## Parameters

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

## Examples

The following 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 <a href="#">"FV" on page 63</a> .
<code>Round(Total_Price, 2)</code>	Rounds off the value of <code>Total_Price</code> to two decimal places.

# Sum

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

**Note:** FormCalc follows the IEEE-754 international standard when handling floating point numeric values. For more information, see [“Number literals” on page 8](#).

## Syntax

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

## Parameters

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

## Examples

The following are examples of using the Sum function:

Expression	Returns
<code>Sum(2, 4, 6, 8)</code>	20
<code>Sum(-2, 4, -6, 8)</code>	4
<code>Sum(4, 16, "abc", 19)</code>	39
<code>Sum(Amount[2], Amount[5])</code>	Totals the third and sixth occurrences of <code>Amount</code> .
<code>Sum(Round(20/3, 2), Max(Amount[*]), Min(Amount[*]))</code>	Totals the value of 20/3 rounded to two decimal places, as well as the largest and smallest non-null occurrences of <code>Amount</code> .  See also <a href="#">“Max” on page 36</a> , <a href="#">“Min” on page 37</a> , and <a href="#">“Round” on page 39</a> .



Functions in this section deal specifically with creating and manipulating date and time values.

### Functions

- ["Date" on page 47](#)
- ["Date2Num" on page 48](#)
- ["DateFmt" on page 49](#)
- ["IsoDate2Num" on page 50](#)
- ["IsoTime2Num" on page 51](#)
- ["LocalDateFmt" on page 52](#)
- ["LocalTimeFmt" on page 53](#)
- ["Num2Date" on page 54](#)
- ["Num2GMTime" on page 55](#)
- ["Num2Time" on page 56](#)
- ["Time" on page 57](#)
- ["Time2Num" on page 58](#)
- ["TimeFmt" on page 59](#)

# Structuring dates and times

## Locales

A locale is a standard term used when developing international standards to identify a particular nation (language and/or country). For the purposes of FormCalc, a locale defines the format of dates and times relevant to a specific nation or region, enabling end-users to use the date and time formats to which they are accustomed.

Each locale is comprised of a unique string of characters or locale identifier. The composition of these strings is controlled by the international standards organization (ISO) Internet Engineering Task Force (IETF), a working group of the Internet Society ([www.isoc.org](http://www.isoc.org)).

Locales are identified by a language code and/or a country code. The following table lists valid locales for this release of Designer:

Identifier	Locale
de_DE	German (Germany)
en_AU	English (Australia)
en_CA	English (Canada)
en_GB	English (Great Britain)
en_US	English (United States)
fr_CA	French (Canada)
fr_FR	French (France)
ja_JP	Japanese (Japan)
ko_KR	Korean (South Korea)
zh_CN	Simplified Chinese (China)

Usually, both elements of a locale are important. For example, the names of weekdays and months in English for Canada and in Great Britain are formatted identically, but dates are formatted differently. So, specifying an English language locale is not enough. Conversely, specifying only a country as the locale is not enough. For example, Canada has different date formats for English and French.

In general, every application operates in an environment where a locale is present. This locale is known as the ambient locale. In some circumstances, an application might operate on a system, or within an environment, where a locale is not present. In these rare cases, the ambient locale is set to a default of English United States (en-US). This locale is known as a default locale.

## Epoch

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

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 1 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 may be returned to users in time zones east of GM T.

## Date formats

A date format is a shorthand specification of how a date appears. It consists of various punctuation marks and symbols representing the formatting that the date must use. The following are examples of date formats:

Date format	Example
MM/DD/YY	11/11/78
DD/MM/YY	25/7/85
MMMM DD, YYYY	March 10, 1964

The format of dates is governed by an ISO. Each country or region specifies its own date formats. There are five general categories of date formats: default, short, medium, long, and full. The table below gives some examples of different date formats from different locales for each of the five categories.

Date format category	Locale identifier	Date format	Example
Default (Typically follows the medium format definition, but not always.)	en_US	MMM D, YYYY	Feb 10, 1970
Short (Numeric)	en_GB	M/D/YY	8/18/92 8/4/92
Medium (0 padded 2 digit month value)	fr_CA	YY/MM/DD	99/06/26
Long (Full month names)	de_DE	D. MMMM YYYY	17. Juni 1989
Full (Includes the weekday name)	fr_FR	EEEE D' MMMM YYYY	Lundi 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 are 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. Each nation specifies the form of its default, short, medium, long, and full time formats. The locale is responsible for identifying the format of times that conform to the standards of that nation. The table below gives some examples of different date formats from different locales for each of the five categories.

Time format category	Locale identifier	Time format	Example
Default (Typically follows the medium format definition for the locale, but not always.)	en_US	h:mm:ss a	7:45:15 a
Short (24 hour system)	en_GB	HH:mm	14:13
Medium (24 hour system and a value for seconds.)	fr_CA	HH:mm:ss	12:15:50
Long (24 hour system, value for seconds, and the hour differential between the locale and Greenwich Mean Time.)	de_DE	HH:mm:ss z	14:13:13 -04:00
Full (Includes the weekday name)	fr_FR	HH' h 'mm z	14 h 13 -0400

## Date and time picture formats

The following symbols must be used to create date and time patterns for date/time fields.

**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.

Date symbol	Description
D	1 or 2 digit (1-31) day of the month
DD	Zero-padded 2 digit (01-31) day of the month
J	1, 2, or 3 digit (1-366) day of the year
JJJ	Zero-padded, three-digit (001-366) day of the year
M	One- or two-digit (1-12) month of the year
MM	Zero-padded, two-digit (01-12) month of the year
MMM	Abbreviated month name
MMMM	Full month name
E	One-digit (1-7) day of the week, where (1=Sunday)
EEE	Abbreviated weekday name
EEEE	Full weekday name
YY	Two-digit year, where 00=2000, 29=2029, 30=1930, and 99=1999
YYYY	Four-digit year
G	Era name (BC or 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
WW	Two-digit (01-53) ISO-8601 week of the year, where week 1 is the week containing January 4

Time symbols	Description
h	One- or two-digit (1-12) hour of the meridian (AM/PM)
hh	Zero-padded 2 digit (01-12) hour of the meridian (AM/PM)
k	One- or two-digit (0-11) hour of the meridian (AM/PM)
kk	Two-digit (00-11) hour of the meridian (AM/PM)
H	One- or two-digit (0-23) hour of the day
HH	Zero-padded, two-digit (00-23) hour of the day
K	One- or two-digit (1-24) hour of the day
KK	Zero-padded, two-digit (01-24) hour of the day
M	One- or two-digit (0-59) minute of the hour
MM	Zero-padded, two-digit (00-59) minute of the hour
S	One- or two-digit (0-59) second of the minute
SS	Zero-padded, two-digit (00-59) second of the minute

Time symbols	Description
FFF	Three- digit (000-999) thousandth of the second
A	Meridian (AM or PM)
z	ISO-8601 time-zone format (for example, z , +0500, -0030, -01, +0100)
z z	Alternative ISO-8601 time-zone format (for example, z , +05:00, -00:30, -01, +01:00)
Z	Abbreviated time-zone name, for example, GMT, GMT+05:00, GMT-00:30, EST, PDT

**Note:** The `h` and `H` patterns reflect 12 and 24-hour clocks respectively.

### Reserved symbols

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

?	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.

# Date

Returns the current system date as the number of days since the [epoch](#).

## Syntax

Date ( )

## Parameters

None

## Examples

The following is an example of using the Date function:

Expression	Returns
Date ( )	37875 (the number of days from the <a href="#">epoch</a> 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 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 examples below illustrate 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
<i>n</i> (optional)	An integer identifying the locale-specific time format style as follows: <ul style="list-style-type: none"><li>• 0 (Default style)</li><li>• 1 (Short style)</li><li>• 2 (Medium style)</li><li>• 3 (Long style)</li><li>• 4 (Full style)</li></ul> If <i>n</i> is omitted (or is invalid), the default style value 0 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.

## Examples

The following are examples of using the DateFmt function:

Expression	Returns
<code>DateFmt ()</code>	MMM D, YYYY This is the default date format.
<code>DateFmt (1)</code>	M/D/YY
<code>DateFmt (2, "fr_CA")</code>	YY-MM-DD
<code>DateFmt (3, "de_DE")</code>	D. MMMM YYYY
<code>DateFmt (4, "fr_FR")</code>	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
<i>d</i>	A valid date string.

### Examples

The following 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
<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
<i>d</i>	A valid time string.

### Examples

The following 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
<i>n</i> ( <i>optional</i> )	An integer identifying the locale-specific time format style as follows: <ul style="list-style-type: none"><li>• 0 (Default style)</li><li>• 1 (Short style)</li><li>• 2 (Medium style)</li><li>• 3 (Long style)</li><li>• 4 (Full style)</li></ul> If <i>n</i> is omitted (or is invalid), the default style value 0 is used.
<i>k</i> ( <i>optional</i> )	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.

### Examples

The following are examples of the LocalDateFmt function:

Expression	Returns
LocalDateFmt (1, "de_DE")	tt.MM.uu
LocalDateFmt (2, "fr_CA")	aa-nn-jj
LocalDateFmt (3, "de_CH")	t. MMMM uuuu
LocalDateFmt (4, "fr_FR")	EEEE j nnnn aaaa

# LocalTimeFmt

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

## Syntax

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

## Parameters

Parameter	Description
<i>n</i> (Optional)	An integer identifying the locale-specific time format style as follows: <ul style="list-style-type: none"><li>• 0 (Default style)</li><li>• 1 (Short style)</li><li>• 2 (Medium style)</li><li>• 3 (Long style)</li><li>• 4 (Full style)</li></ul> If <i>n</i> is omitted (or is invalid), the default style value 0 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.

## Examples

The following 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
<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 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 examples below illustrate using the Num2Date function:

Expression	Returns
<code>Num2Date(1, "DD/MM/YYYY")</code>	<code>01/01/1900</code>
<code>Num2Date(35139, "DD-MMM-YYYY", "de_DE")</code>	<code>16-Mrz-1996</code>
<code>Num2Date(Date2Num("Mar 15, 2000") - Date2Num("98-03-15", "YY-MM-DD", "fr_CA"))</code>	<code>Jan 1, 1902</code>

# 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 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 examples below illustrate using the Num2GMTTime function:

Expression	Returns
Num2GMTTime(1, "HH:MM:SS")	00:00:00
Num2GMTTime(65593001, "HH:MM:SS Z")	18:13:13 GMT
Num2GMTTime(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
<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 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 examples below illustrate using the Num2Time function:

Expression	Returns
<code>Num2Time(1, "HH:MM:SS")</code>	<code>00:00:00</code> in Greenwich, England and <code>09:00:00</code> in Tokyo.
<code>Num2Time(65593001, "HH:MM:SS Z")</code>	<code>13:13:13 EST</code> in Boston, U.S.
<code>Num2Time(65593001, "HH:MM:SS Z", "de_DE")</code>	<code>13:13:13 GMT-05:00</code> to a German-Swiss user in Boston, U.S.
<code>Num2Time(43993001, TimeFmt(4, "de_DE"), "de_DE")</code>	<code>13.13 Uhr GMT+01:00</code> to a user in Zurich, Austria.
<code>Num2Time(43993001, "HH:MM:SSzz")</code>	<code>13:13+01:00</code> 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 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 Time (ET) 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 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 examples below 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>	76393001 to a user in California on Pacific Standard Time, and 76033001 when 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>	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.
<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
<i>n</i> (Optional)	An integer identifying the locale-specific time format style as follows: <ul style="list-style-type: none"><li>• 0 (Default style)</li><li>• 1 (Short style)</li><li>• 2 (Medium style)</li><li>• 3 (Long style)</li><li>• 4 (Full style)</li></ul> If you do not include a value for <i>n</i> , or if <i>n</i> is invalid, the function uses the default style value.
<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.

## Examples

The following are examples of using the TimeFmt function:

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

# 6

## Financial Functions

---

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

### Functions

- ["Apr" on page 61](#)
- ["CTerm" on page 62](#)
- ["FV" on page 63](#)
- ["IPmt" on page 64](#)
- ["NPV" on page 65](#)
- ["Pmt" on page 66](#)
- ["PPmt" on page 67](#)
- ["PV" on page 68](#)
- ["Rate" on page 69](#)
- ["Term" on page 70](#)

# 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
<i>n1</i>	A numeric value or expression representing the principal amount of the loan.
<i>n2</i>	A numeric value or expression representing the payment amount on the loan.
<i>n3</i>	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 8](#).

## Examples

The following are examples of using the Apr function:

Expression	Returns
<code>Apr(35000, 269.50, 360)</code>	0.08515404566 for a \$35,000 loan repaid at \$269.50 a month for 30 years.
<code>Apr(210000 * 0.75, 850 + 110, 25 * 26)</code>	0.07161332404
<code>Apr(-20000, 250, 120)</code>	Error
<code>Apr(P_Value, Payment, Time)</code>	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
<i>n1</i>	A numeric value or expression representing the interest rate per period.
<i>n2</i>	A numeric value or expression representing the future value of the investment.
<i>n3</i>	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 8](#).

### Examples

The following are examples of using the CTerm function:

Expression	Returns
<code>CTerm(0.02, 1000, 100)</code>	116.2767474515
<code>CTerm(0.10, 500000, 12000)</code>	39.13224648502
<code>CTerm(0.0275 + 0.0025, 1000000, 55000 * 0.10)</code>	176.02226044975
<code>CTerm(Int_Rate, Target_Amount, P_Value)</code>	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
<i>n1</i>	A numeric value or expression representing the payment amount.
<i>n2</i>	A numeric value or expression representing the interest per period of the investment.
<i>n3</i>	A numeric value or expression representing the total number of payment periods.

The function returns an error if either of the following 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 8](#).

## Examples

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

The function returns an error if either of the following 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 8](#).

## Examples

The following are examples of using 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
<i>n1</i>	A numeric value or expression representing the discount rate over a single period.
<i>n2</i>	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 <i>n2</i> 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 8](#).

### Examples

The following are examples of using the NPV function:

Expression	Returns
<code>NPV(0.065, 5000)</code>	4694.83568075117, which is the net present value of an investment earning 6.5% per year that will generate \$5000.
<code>NPV(0.10, 500, 1500, 4000, 10000)</code>	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.
<code>NPV(0.0275 / 12, 50, 60, 40, 100, 25)</code>	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
<i>n1</i>	A numeric value or expression representing the principal amount of the loan.
<i>n2</i>	A numeric value or expression representing the interest rate per period of the investment.
<i>n3</i>	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 8](#).

## Examples

The following are examples of using 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
<i>n1</i>	A numeric value or expression representing the principal amount of the loan.
<i>n2</i>	A numeric value or expression representing the annual interest rate.
<i>n3</i>	A numeric value or expression representing the amount of the monthly payment.
<i>n4</i>	A numeric value or expression representing the first month in which a payment will be made.
<i>n5</i>	A numeric value or expression representing the number of months for which to calculate.

The function returns an error if either of the following 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 8](#).

### Examples

The following are examples of using 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
<i>n1</i>	A numeric value or expression representing the payment amount.
<i>n2</i>	A numeric value or expression representing the interest per period of the investment.
<i>n3</i>	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 8](#).

## Examples

The following are examples of using 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
<i>n1</i>	A numeric value or expression representing the future value of the investment.
<i>n2</i>	A numeric value or expression representing the present value of the investment.
<i>n3</i>	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 8](#).

## Examples

The following are examples of using 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.
<code>Rate(10000, 0.25 * 5000, 4 * 12)</code>	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.
<code>Rate(Target_Value, Pres_Value[*], Term * 12)</code>	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
<i>n1</i>	A numeric value or expression representing the payment amount made at the end of each period.
<i>n2</i>	A numeric value or expression representing the interest rate per period of the investment.
<i>n3</i>	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 8](#).

## Examples

The following are examples of using 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.

Expression	Returns
<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.

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

## Functions

- ["Choose" on page 73](#)
- ["Exists" on page 74](#)
- ["HasValue" on page 75](#)
- ["Oneof" on page 76](#)
- ["Within" on page 77](#)



# 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 is true: <ul style="list-style-type: none"><li>• <i>n</i> is less than 1.</li><li>• <i>n</i> is greater than the number of items in the set.</li></ul> If <i>n</i> is null, the function returns null.
<i>s1</i>	The first value in the set of values.
<i>s2 (Optional)</i>	Additional values in the set.

## Examples

The following are examples of using 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 set <i>Items</i> that corresponds to the position defined by the first occurrence of <i>Item_Num</i> .
<code>Choose(20/3, "A", "B", "C", "D", "E", "F", "G", "H")</code>	F

# Exists

Determines whether the given parameter is an accessor to an existing object.

## Syntax

`Exists(v)`

## Parameters

Parameter	Description
<code>v</code>	A valid accessor value. If <code>v</code> is not an accessor, the function returns false (0).

## Examples

The following are examples of using 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 an accessor.
<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 an accessor with a non-null, non-empty, or non-blank value.

## Syntax

HasValue (v)

## Parameters

Parameter	Description
v	A valid accessor value. If v is not an accessor, the function returns false (0).

## Examples

The following are examples of using 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
<i>s1</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>s1</i> down to the nearest whole value.
<i>s2</i>	The first value in the set of values.
<i>s3 (Optional)</i>	Additional values in the set.

## Examples

The following are examples of using the Oneof function:

Expression	Returns
<code>Oneof(3, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)</code>	True (1)
<code>Oneof("John", "Bill", "Gary", "Joan", "John", "Lisa")</code>	True (1)
<code>Oneof(3, 1, 25)</code>	False (0)
<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
<i>s1</i>	The value to test for. If <i>s1</i> is a number, the ordering comparison is numeric. If <i>s1</i> is not a number, the ordering comparison uses the collating sequence for the current locale. For more information, see <a href="#">“Locales” on page 42</a> . If <i>s1</i> is null, the function returns null.
<i>s2</i>	The lower bound of the test range.
<i>s3</i>	The upper bound of the test range.

## Examples

The following are examples of using 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.

# 8

## Miscellaneous Functions

---

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

### Functions

- ["Eval" on page 79](#)
- ["Null" on page 80](#)
- ["Ref" on page 81](#)
- ["UnitType" on page 82](#)
- ["UnitValue" on page 83](#)

# Eval

Returns the value of a given form calculation.

## Syntax

`Eval (s)`

## Parameters

Parameter	Description
<i>s</i>	<p>A valid string representing an expression or list of expressions.</p> <p><b>Note:</b> The Eval function 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, the Eval function does not recognize <i>s</i>, and so returns an error. Any subsequent functions that make reference to the variable <i>s</i> also fail.</p>

## Examples

The following are examples of using 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 really means no value.

## Definition

`Null()`

## Parameters

None

## Examples

The following are examples of using the Null function:

Expression	Returns
<code>Null()</code>	<i>null</i>
<code>Null() + 5</code>	5
<code>Quantity = Null()</code>	Assigns <i>null</i> to the object Quantity.
<code>Concat("ABC", Null(), "DEF")</code>	ABCDEF See also <a href="#">“Concat” on page 86</a> .



# Ref

Returns a reference to an existing object.

## Definition

Ref (v)

## Parameters

Parameters	Description
v	A valid string representing an accessor, reference, method, or function. <b>Note:</b> 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 are examples of using 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
<i>s</i>	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"><li>• in, inches</li><li>• mm, millimeters</li><li>• cm, centimeters</li><li>• pt, picas, points</li><li>• mp, millipoints</li></ul> If <i>s</i> is invalid, the function returns in.

## Examples

The following are examples of using the UnitType function:

Expression	Results
<code>UnitType("36 in")</code>	in
<code>UnitType("2.54centimeters")</code>	cm
<code>UnitType("picas")</code>	pt
<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 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
<i>s1</i>	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"><li>• in, inches</li><li>• mm, millimeters</li><li>• cm, centimeters</li><li>• pt, picas, points</li><li>• mp, millipoints</li></ul>
<i>s2</i> (optional)	A string containing a valid unit of measurement. The function converts the unitspan specified in <i>s1</i> to this new unit of measurement.  If you do not include a value for <i>s2</i> , the function uses the unit of measurement specified in <i>s1</i> . If <i>s2</i> is invalid, the function converts <i>s1</i> into inches.

## Examples

The following are examples of using 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 <i>Size</i> converted into millipoints.
UnitValue("5.08cm", "kilograms")	2

Functions in this section deal with the manipulation, evaluation, and creation of string values.

## Functions

- ["At" on page 85](#)
- ["Concat" on page 86](#)
- ["Decode" on page 87](#)
- ["Encode" on page 88](#)
- ["Format" on page 89](#)
- ["Left" on page 90](#)
- ["Len" on page 91](#)
- ["Lower" on page 92](#)
- ["Ltrim" on page 93](#)
- ["Parse" on page 94](#)
- ["Replace" on page 95](#)
- ["Right" on page 96](#)
- ["Rtrim" on page 97](#)
- ["Space" on page 98](#)
- ["Str" on page 99](#)
- ["Stuff" on page 100](#)
- ["Substr" on page 101](#)
- ["Uuid" on page 102](#)
- ["Upper" on page 103](#)
- ["WordNum" on page 104](#)

# 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 are examples of using 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 Cust_Info. See also <a href="#">"Ltrim" on page 93</a> .

## 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 (Optional)</i>	Additional strings to append to the set.

### Examples

The following are examples of using the Concat function:

Expression	Returns
<code>Concat ("ABC", "DEF")</code>	ABCDEF
<code>Concat ("Tony", Space(1), "Blue")</code>	Tony Blue See also <a href="#">"Space" on page 98</a> .
<code>Concat ("You owe ", WordNum(1154.67, 2), ".")</code>	You owe One Thousand One Hundred Fifty-four Dollars And Sixty-seven Cents. See also <a href="#">"WordNum" on page 104</a> .

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

### Examples

The following are examples of using the Decode function:

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

# Encode

Returns the encoded version of a given string.

## Syntax

`Encode ( s1 [, s2 ] )`

## Parameters

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

## Examples

The following are examples of using the Encode function:

Expression	Returns
<code>Encode ("hello, world!", "url")</code>	<code>%22hello,%20world!%22</code>
<code>Encode ("ÃÄÅæ", "html")</code>	<code>&amp;#xc1;&amp;#xc2;&amp;#xc3;&amp;#xc4;&amp;#xc5;&amp;#xc6;</code>



## Format

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

### Syntax

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

### Parameters

Parameter	Description
<i>s1</i>	The picture format string, which may be a locale-sensitive picture clause. See <a href="#">"Locales" on page 42</a> .
<i>s2</i>	<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"> <li>• YYYY[MM[DD]]</li> <li>• YYYY[-MM[-DD]]</li> </ul> <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"> <li>• HH[MM[SS[.FFF][z]]]</li> <li>• HH[MM[SS[.FFF][+HH[MM]]]]</li> <li>• HH[MM[SS[.FFF][-HH[MM]]]]</li> <li>• HH[:MM[:SS[.FFF][z]]]</li> <li>• HH[:MM[:SS[.FFF][-HH[:MM]]]]</li> <li>• HH[:MM[:SS[.FFF][+HH[:MM]]]]</li> </ul> <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>
<i>s3 (Optional)</i>	Additional source data to format.

### Examples

The following are examples of using 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. 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
<i>s</i>	The string to extract from.
<i>n</i>	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 are examples of using 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 of Telephone.
<code>Left (Rtrim (Last_Name), 3)</code>	The first three characters of Last_Name. See also <a href="#">"Rtrim" on page 97</a> .

# Len

Returns the number of characters in a given string.

## Syntax

Len (*s*)

## Parameters

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

## Examples

The following are examples of using the Len function:

Expression	Returns
Len ("ABDCEFGH")	8
Len (4)	1
Len (Str (4.532, 6, 4))	6 See also <a href="#">"Str" on page 99</a> .
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
<i>s</i>	The string to convert.
<i>k</i> (Optional)	<p>A string representing a valid locale. If you do not include a value for <i>k</i>, the function uses the ambient locale.</p> <p>See also <a href="#">"Locales" on page 42</a>.</p> <p><b>Note:</b> Characters outside the Latin1 subrange of the Unicode 2.1 character set are never converted.</p>

### Examples

The following are examples of using the Lower function:

Expression	Returns
<code>Lower("ABC")</code>	abc
<code>Lower("21 Main St.")</code>	21 main st.
<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
<i>s</i>	The string to trim.

## Examples

The following are examples of using the Ltrim function:

Expression	Returns
<code>Ltrim("        ABCD")</code>	"ABCD"
<code>Ltrim(Rtrim("        Tony Blue        "))</code>	"Tony Blue" See also <a href="#">"Rtrim" on page 97</a> .
<code>Ltrim(Address[0])</code>	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:

- 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
<i>s1</i>	A valid date or time picture format string. For more information on date and time formats, see <a href="#">“Date and Time Functions” on page 41</a> .
<i>s2</i>	The string data to parse.

## Examples

The following are examples of using 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>	1234567.89 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
<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 are examples of using 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
<i>s</i>	The string to extract.
<i>n</i>	The number of characters to extract. If <i>n</i> is greater than the length of the string, the function returns the whole string. If <i>n</i> is 0 or less, the function returns an empty string.

## Examples

The following are examples of using 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 <a href="#">"Rtrim" on page 97</a> .



## 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 are examples of using the Rtrim function:

Expression	Returns
<code>Rtrim("ABCD ")</code>	"ABCD"
<code>Rtrim("Tony Blue ")</code>	"Tony Blue"
<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
<i>n</i>	The number of blank spaces.

## Examples

The following are examples of using 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 <a href="#">"Max" on page 36</a> .
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
<i>n1</i>	The number to convert.
<i>n2</i> (Optional)	The maximum width of the string. If you do not include a value for <i>n2</i> , the function uses a value of 10 as the default width.  If the resulting string is longer than <i>n2</i> , the function returns a string of * (asterisk) characters of the width specified by <i>n2</i> .
<i>n3</i> (Optional)	The number of digits to appear after the decimal point. If you do not include a value for <i>n3</i> , the function uses 0 as the default precision.

## Examples

The following are examples of using 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 <i>Amount</i> to a six-character string with two decimal places.  See also <a href="#">"Max" on page 36</a> .

# Stuff

Inserts a string into another string.

## Syntax

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

## Parameters

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

## Examples

The following are examples of using 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 <code>Street</code> onto the end of the first occurrence of <code>Address</code> . See also <a href="#">"Len" on page 91</a> .
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
<i>s1</i>	The source string.
<i>n1</i>	The position in string <i>s1</i> to start extracting. If <i>n1</i> is less than one, the function assumes the first character position. If <i>n1</i> is greater than length of <i>s1</i> , the function assumes the last character position.
<i>n2</i>	The number of characters to extract. If <i>n2</i> is less than or equal to 0, FormCalc returns an empty string. If <i>n1</i> + <i>n2</i> is greater than the length of <i>s1</i> , the function returns the substring starting at position <i>n1</i> to the end of <i>s1</i> .

## Examples

The following are examples of using the Substr function:

Expression	Returns
<code>Substr("ABCDEFGH", 3, 4)</code>	CDEF
<code>Substr(3214, 2, 1)</code>	2
<code>Substr&gt;Last_Name[0], 1, 3)</code>	Returns the first three characters from the first occurrence of <code>Last_Name</code> .
<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
<i>n</i>	A number identifying the format of the UUID string. Valid numbers are: <ul style="list-style-type: none"><li>0 (default value): UUID string only contains hex octets.</li><li>1: UUID string contains dash characters separating the sequences of hex octets at fixed positions.</li></ul> If you do not include a value for <i>n</i> , the function uses the default value.

## Examples

The following 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
<i>s</i>	The string to convert.
<i>k</i> (Optional)	<p>A string representing a valid locale. If you do not include a value for <i>k</i>, the ambient locale is used.</p> <p>See also <a href="#">"Locales" on page 42</a>.</p> <p><b>Note:</b> Characters outside the Latin1 subrange of the Unicode 2.1 character set are never converted.</p>

## Examples

The following are examples of using 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>	<p>The number to convert.</p> <p>If any of the following statements is true, the function returns * (asterisk) characters to indicate an error:</p> <ul style="list-style-type: none"><li>• <i>n1</i> is not a number.</li><li>• The integral value of <i>n1</i> is negative.</li><li>• The integral value of <i>n1</i> is greater than 922,337,203,685,477,550.</li></ul>
<i>n2</i> (Optional)	<p>A number identifying the formatting option. Valid numbers are:</p> <ul style="list-style-type: none"><li>• 0 (default value): The number is converted into text representing the simple number.</li><li>• 1: The number is converted into text representing the monetary value with no fractional digits.</li><li>• 2: The number is converted into text representing the monetary value with fractional digits.</li></ul> <p>If you do not include a value for <i>n2</i>, the function uses the default value (0).</p>
<i>k</i> (Optional)	<p>A string representing a valid locale. If you do not include a value for <i>k</i>, the function uses the ambient locale.</p> <p>See also <a href="#">“Locales” on page 42</a>.</p> <p><b>Note:</b> As of this release, it is not possible to specify a locale identifier other than English for this function.</p>



## Examples

The following are examples of using 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 <code>Amount</code> as the conversion number.

# 10

## URL Functions

---

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

### Functions

- ["Get" on page 107](#)
- ["Post" on page 108](#)
- ["Put" on page 110](#)

# Get

Downloads the contents of the given URL.

## Syntax

Get (*s*)

## Parameters

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

## Examples

The following are examples of using 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 a SQL query to the specified web site.

# Post

Posts the given data to the specified URL.

## Syntax

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

## Parameters

Parameter	Description
<i>s1</i>	The URL to post to.
<i>s2</i>	The data to post. If the function is unable to post the data, it returns an error.
<i>s3 (Optional)</i>	A string containing the content type of the data to post. Valid content types include: <ul style="list-style-type: none"> <li>• application/octet-stream (default value)</li> <li>• text/html</li> <li>• text/xml</li> <li>• text/plain</li> <li>• multipart/form-data</li> <li>• application/x-www-form-urlencoded</li> <li>• Any other valid MIME type</li> </ul> If you do not include a value for <i>s3</i> , the function sets the content type to the default value. The application is responsible for ensuring that the data to post uses the correct format according to the specified content type.
<i>s4 (Optional)</i>	A string containing the name of the code page used to encode the data. Valid code page names include: <ul style="list-style-type: none"> <li>• UTF-8 (default value)</li> <li>• UTF-16</li> <li>• ISO-8859-1</li> <li>• Any character encoding listed by the Internet Assigned Numbers Authority (IANA)</li> </ul> If you do not include a value for <i>s4</i> , the function sets the code page to the default value. The application is responsible for ensuring that encoding of the data to post matches the specified code page.
<i>s5 (Optional)</i>	A string containing any additional HTTP headers to be included with the posting of the data. If you do not include a value for <i>s5</i> , 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 are examples of using the Post function:

Expression	Returns
<code>Post("http://tools_build/scripts/jfecho.cgi", "user=joe&amp;passwd=xxxxx&amp;date=27/08/2002", "application/x-www-form-urlencoded")</code>	Posts some url encoded login data to a server and returns that server's acknowledgement page.
<code>Post("http://www.nanonull.com/TimeService/ TimeService.asmx/getLocalTime", "&lt;?xml version='1.0' encoding='UTF-8'?&gt;&lt;soap:Envelope&gt;&lt;soap:Body&gt; &lt;getLocalTime/&gt;&lt;/soap:Body&gt; &lt;/soap:Envelope&gt;", "text/xml", "utf-8", "http://www.Nanonull.com/TimeService/getLocalTime")</code>	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.

## Syntax

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

## Parameters

Parameter	Description
<i>s1</i>	The URL to upload.
<i>s2</i>	The data to upload. If the function is unable to upload the data, it returns an error.
<i>s3</i> (Optional)	A string containing the name of the code page used to encode the data. Valid code page names include: <ul style="list-style-type: none"><li>• UTF-8 (default value)</li><li>• UTF-16</li><li>• ISO8859-1</li><li>• Any character encoding listed by the Internet Assigned Numbers Authority (IANA)</li></ul> If you do not include a value for <i>s3</i> , the function sets the code page to the default value. The application is responsible for ensuring that encoding of the data to upload matches the specified code page.

## Examples

The following is an example of using the Put function:

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

# Index

## A

- Abs (Arithmetic function) 31
- absolute value 31
- accessors 20, 74, 75
- alphabetical function list 26
- ambient locale 42
- analyzing a string 94
- annual percentage rate 61
- Apr (Financial function) 61
- Arithmetic functions
  - Abs 31
  - Avg 32
  - Ceil 33
  - Count 34
  - Floor 35
  - Max 36
  - Min 37
  - Mod 38
  - Round 39
  - Sum 40
- array referencing 23
- assignment expressions 16
- At (String function) 85
- Avg (Arithmetic function) 32

## B

- blank spaces 98

## C

- carriage return 13
- Ceil (Arithmetic function) 33
- character position within a string 85
- Choose (Logical function) 73
- code pages 108, 110
- comments 11
- compound
  - interest rate 69
  - picture formats 89
- compounding term 62
- Concat (String function) 86
- conditional statements 19
- constant values 8
- converting
  - a time string to a number 58
  - characters to lowercase 92
  - characters to uppercase 103
  - numbers to a string 99
  - numbers to text 104
  - operands 15
- Count (Arithmetic function) 34
- counting characters in a string 91
- CTerm (Financial function) 62

- current system date 47

- current system time 57

## D

- Date (Date and Time function) 47
- Date and Time functions
  - Date 47
  - Date2Num 48
  - DateFmt 49
  - IsoDate2Num 50
  - IsoTime2Num 51
  - LocalDateFmt 52
  - LocalTimeFmt 53
  - Num2Date 54
  - Num2GMTTime 55
  - Num2Time 56
  - Time 57
  - Time2Num 58
  - TimeFmt 59
- date format 43
  - string 49
  - string, localized 52
- date picture format 89, 94
- date picture formats 44
- date string, from number 54
- Date2Num (Date and Time function) 48
- DateFmt (Date and Time function) 49
- date-time picture formats 94
- days since the epoch 47, 48, 50
- Decode (String function) 87
- default locale 42
- downloading URL contents 107

## E

- empty string 9, 98
- Encode (String function) 88
- epoch 43
- equality expressions 17
- escape sequence 10
- Eval (Miscellaneous function) 79
- Exists (Logical function) 74
- expressions 13
  - assignment 16
  - equality 17
  - if 19
  - inequality 17
  - logical AND 16
  - logical OR 16
  - relational 18
  - simple 14
  - unary 17
- extracting from a string 90, 96, 101

## F

- Financial functions
  - Apr 61
  - CTerm 62
  - FV 63
  - IPmt 64
  - NPV 65
  - Pmt 66
  - PPmt 67
  - PV 68
  - Rate 69
  - Term 70
- Floor (Arithmetic function) 35
- form feed 13
- Format (String function) 89
- FormCalc variables 12
- function calls 25
- functions, alphabetical list 26
- future value 63
- FV (Financial function) 63

## G

- Get (URL function) 107
- GMT time string, from number 55

## H

- HasValue (Logical function) 75
- horizontal tab 13

## I

- identification, unique 102
- identifier, locale 42
- identifiers 12
- if expressions 19
- inequality expressions 17
- inserting a string into another string 100
- Institute of Electrical and Electronics Engineers (IEEE) 8
- interest paid on a loan 64
- interest rate 69
- investment
  - periods required 70
  - present value 68
- IPmt (Financial function) 64
- ISO date string 94
- ISO date-time string 89
- ISO time string 94
- ISO-8859-1 108, 110
- IsoDate2Num (Date and Time function) 50
- IsoTime2Num (Date and Time function) 51

## J

- joining strings 86

## K

- keywords 11

## L

- Left (String function) 90

- Len (String function) 91
- line feed 13
- line terminators 13
- list of functions, alphabetical 26
- Literals 8
  - Number literals 8
  - String literals 9
- loan payments 66
- loan principal paid 67
- LocalDateFmt (Date and Time function) 52
- locale 42
- localized date format string 52
- localized time format string 53
- LocalTimeFmt (Date and Time function) 53
- logical AND expressions 16
- Logical functions
  - Choose 73
  - Exists 74
  - HasValue 75
  - Oneof 76
  - Within 77
- logical OR expressions 16
- Lower (String function) 92
- Ltrim (String function) 93

## M

- Max (Arithmetic function) 36
- milliseconds since the epoch 51
- MIME types 108
- Min (Arithmetic function) 37
- Miscellaneous functions
  - Eval 79
  - Ref 81
  - UnitType 82
  - UnitValue 83
- Mod (Arithmetic function) 38
- modulus 38

## N

- net present value 65
- NPV (Financial function) 65
- null values 34
- Num2Date (Date and Time function) 54
- Num2GMTTime (Date and Time function) 55
- Num2Time (Date and Time function) 56
- Number literals 8
- number of
  - days since the epoch 47, 48, 50
  - milliseconds since the epoch 51
  - periods 70
- numbers to text 104
- numeric picture format 94
- numeric picture formats 89

## O

- Oneof (Logical function) 76
- operands, promoting 15
- operators 10, 14



## P

- Parse (String function) 94
- payments on a loan 66
- picture format 89, 94
- picture formats, date and time 44
- Pmt (Financial function) 66
- Post (URL function) 108
- posting to a URL 108
- PPmt (Financial function) 67
- present value 68
- principal paid on a loan 67
- Put (URL function) 110
- PV (Financial function) 68

## R

- Rate (Financial function) 69
- Ref (Miscellaneous function) 81
- relational expressions 18
- remainder 38
- removing white space 93, 97
- Replace (String function) 95
- Right (String function) 96
- Round (Arithmetic function) 39
- Rtrim (String function) 97

## S

- security 102
- simple expressions 14
- Space (String function) 98
- starting character position 85
- Str (String function) 99
- string
  - analyzing 94
  - blank 98
  - converting numbers 99
  - extracting a section 101
  - extracting characters from the left 90
  - extracting characters from the right 96
  - formatting 89
  - inserting into another string 100
  - length 91
  - lowercase 92
  - removing white space 93, 97
  - replacing 95
  - uppercase 103
- String functions
  - At 85
  - Concat 86
  - Decode 87
  - Encode 88
  - Format 89
  - Left 90
  - Len 91
  - Lower 92
  - Ltrim 93
  - Parse 94
  - Replace 95

## String functions (Continued)

- Right 96
- Rtrim 97
- Space 98
- Str 99
- Stuff 100
- Substr 101
- Upper 103
- Uuid 102
- WordNum 104
- String literals 9
- Stuff (String function) 100
- Substr (String function) 101
- substring 101
- Sum (Arithmetic function) 40
- symbols
  - date pattern 45
  - reserved date/time 46
  - time pattern 45

## T

- Term (Financial function) 70
- terminators, line 13
- text picture formats 89, 94
- Time (Date and Time function) 57
- time format 44
  - string 59
  - string, localized 53
- time picture formats 44, 89, 94
- time since the epoch 57
- time string, from number 56
- Time2Num (Date and Time function) 58
- TimeFmt (Date and Time function) 59

## U

- unary expressions 17
- Unicode escape sequence 10
- unitspan 82, 83
- UnitType (Miscellaneous function) 82
- UnitValue (Miscellaneous function) 83
- Universally Unique Identifier (UUID) 102
- uploading to a URL 110
- Upper (String function) 103
- URL functions
  - Get 107
  - Post 108
  - Put 110
- UTF-16 108, 110
- UTF-8 108, 110
- Uuid (String function) 102

## V

- variables 12
- vertical tab 13

## W

- white space 13, 93, 97
- Within (Logical function) 77
- WordNum (String function) 104