Another efficient and affordable ACT! Add-On by

exponenciel

Reference Guide

Automatic Field Calculations for ACT! 2011 and higher

http://www.exponenciel.com

Table of contents

Operators

Basic Operators:

You may use any of these four basic operators:

+ (plus)
– (minus)
* (multiplied by)
/ (divided by)

You may use parentheses as well.

Examples of calculations:

You may use any combination of the above operators with or without parentheses.

For instance:

([FieldA]+100)*0.05	→	5% of FieldA + 100
[FieldA]+[FieldB]-[FieldC]	→	FieldA + FieldB – FieldC
([FieldA]*125) + ([FieldB]/5)	→	etc.
[FieldA]	→	Copies FieldA to the target field
0	→	Resets target field to 0
[FieldA]+" "+[FieldB]	→	Concatenates FieldA and FieldB w/ a space between

etc.

Operator precedence

Typically the operator precedence (the order in which the operators are evaluated) is */+- which means that:

50*3+1 equals 151 and not 200 (you first multiply then add)

Automatic Field Calculations follows this rule for simple calculations like the one above but for more complicated calculations, **make sure you use parenthesis to help Automatic Field Calculations determine in which order it should proceed with the calculations** (in the example above it would mean writing (50*3)+1 instead of 50*3+1).

Besides the 4 basic operators, you may also use Boolean operators. **Yes** is returned if the comparison is true. **No** if the comparison is false.

- **>, <, >=, <=, <>** allow to compare values.
  Ex: if FieldA = 5 and FieldB = 10,[FieldA]>[FieldB] returns No.

- **NOT** is used to perform a logical negation.
  Ex: if field FieldA = 5 and FieldB = 10, NOT([FieldA]>[FieldB]) returns Yes.

- **AND, OR, XOR** (logical exclusion), **EQV** (logical equivalence), **IMP** (logical implication) may also be used.
  Ex: if field FieldA =5 and 5 FieldB 1=10, ([FieldA]=5)OR([FieldB]=8) returns Yes.

Note: *If you would rather get the numeric equivalent of True and False returned instead of* ***Yes*** *and* ***No****, add a tilde sign ~ in front of your calculation.* ***-1*** *will be returned instead of* ***Yes*** *and* ***0*** *instead of* ***No.***

FieldA = 5 and 51 = 10 and [FieldC]=[FieldA]>[FieldB] then [FieldC]="No"
[FieldC]=~[FieldA]>[FieldB] then [FieldC]="0"

Functions

Automatic Field Calculations can accept a number of different functions. The function names are not case-sensitive.

Text/String functions

Even though not always necessary, it is recommended to always surround strings with quotes.

- **&** is really an operator. It concatenates strings.
  Ex: if FieldA = "Mr." and FieldB= "Huffman" then [FieldA]&[FieldB] returns Mr.Huffman.
  To insert a space use the underscore character as ACT! won't allow a space in the arguments: [FieldA]&"_"&[FieldB] returns "Mr. Huffman".

- **Ucase** will convert to uppercase.
  Ex: if FieldA = "Chicago" then UCase([FieldA]) returns CHICAGO.

- **LCase** will convert to lowercase.
  Ex: if FieldA = "Chicago" then LCase([FieldA]) returns chicago.

- **Fcase** will capitalize every first letter of the words of a string
  Ex: if FieldA=" A1 haRDWARE " then Fcase[FieldA] returns A1 HaRDWARE.

- **FcaseL** will capitalize every first letter of the words of a string and make all other letters lowercase.
  Ex: if FieldA="A1 haRDWARE" then Fcase[FieldA] returns A1 Hardware.

- **Trim** will remove leading and trailing spaces.
  Ex: Trim(" Chicago ") returns Chicago.

- **Ltrim** will remove leading spaces.
  Ex: Ltrim(" Chicago") returns Chicago.

- **Rtrim** will remove trailing spaces.
  Ex: Rtrim("Chicago ") returns Chicago.

- **Len** will return the number of characters in a string.
  Ex: if FieldA = " Chicago" then Len([FieldA]) returns 7.

- **Space(x)** will insert x spaces.
  Ex: space(10) returns " " (10 spaces).

- **String(value)** will force to consider value as a string even if it is numeric or currency.

Ex: String($65.00)&" + taxes" returns "$65.00 + tax" (whereas "$65.00"&" + taxes" would return "65 + taxes").

- **Val** returns a number in a string. It stops at the first character it does not recognize as part of a number.
  Ex: if FieldA = "12C" then Val([FieldA]) returns 12.

- **Left(value;#_of_chars)** returns the first #_of_chars characters from the value, starting from the left
  Ex: if [FieldA]="414-555-1212" then Left([FieldA];3)="414"

- **Right(value;#_of_chars)** returns the first #_of_chars characters from the value, starting from the right
  Ex: if [FieldA]="414-555-1212" then Left([FieldA];8)= "555-1212"

- **Mid(value;start_pos;#_of_chars)** returns #_of_chars from the value, starting from the start_posth character. If #_of_chars is omitted, it returns all characters right of the pos.
  Ex: Ex: if [FieldA]="414-555-1212" then mid([FieldA];5;3)= "555"

- **Replace(value;char(s)_to_replace;replacement)** replaces the character(s) to replace by the replacement.
  Ex: if [FieldA]="24 North Avenue" then Replace([FieldA];"e";"i")= "24 North Avinui"
  Ex: if [FieldA]="24 North Avenue" then Replace([FieldA];"nue";".")= "24 North Ave."

- **ReplaceWord(value;word_to_replace;replacement_word)** replaces the word to replace by the replacement word.
  Ex: if [FieldA]="24 North Avenue" then Replace([FieldA];"Avenue";"Ave.")= "24 North Ave."

- **Initials(value)** returns a string made of the first letters of each word, in uppercase.
  Ex: if [FieldA]="Chris Huffman" then Initials([FieldA])="CH"

- **Contains(start;StringBeingSearched;StringSought;<1>)** returns an integer specifying the start position of the StringSought within the StringBeingSearched. The last argument if used should be one and specifies that the search is not case-sensitive
  Ex: if [FieldA]="Chris Huffman" then Contains(1;"H";[FieldA])=7
  if [FieldA]="Chris Huffman" then Contains(1;"H";[FieldA];1)=2
  If the StringSought is not found, it return 0;

- **Frac2Num(string)** converts a fraction to a decimal.
  Ex: if [FieldA]="1/4" then Frac2Num([FieldA])=.25

Mathematical functions

- **Abs** returns the absolute value of a number.
  Ex: if FieldA = -5.32 then Abs([FieldA]) returns 5.32.

- **Sgn** returns 1 if the number is >0, 0 if the number =0, -1 if the number is <0.
  Ex: if FieldA = -5.32 then Sgn([FieldA]) returns -1.

- **Sqrt** returns the square root of a number.
  Ex: if FieldA=25 then Sqrt([FieldA]) returns 5.

- **Int** returns the integer portion of a number.
  Ex: if FieldA=25.52 then Int([FieldA]) returns 25.

- The **cos, sin, tan, exp, atan** and **log** functions are supported.

- **Round(value;#_of_decimals**) returns a number rounded to a specified number of decimal places.
  Ex: if [FieldA]="123.4875" then Round([FieldA];3)="123.488"

If function

The syntax of the if function is the following:

- **If (logical_test;value_if_true;value_if_false)**
  Ex: if field [FieldB]=1 then If([FieldB]=1;"Yes";"No") returns "Yes"

  The value_if_true and value_if_false arguments are optional.
  Omit them if you don't want the value of the field to be changed.
  Ex: if field [FieldB]=0 then If([FieldB]=1;"Yes";) will not modify the target field.
  Ex: if field [FieldB]=1 then If([FieldB]=1;;"No") will not modify the target field.

  If, on the contrary, you wish to blank out the target field, then use double-quotes.
  Ex: if field [FieldB]=0 then If([FieldB]=1;"Yes";"") will blank out the target field.

Comparison functions

- **Case (reference_value;value1:result1;value2:result2...;else:resultx)** compares value1, value2, ... to reference_value. As soon as it finds a match, it returns the associated value. If no match is found, it returns the value associated with the keyword Else. Note the use of : to associate a value to a result.
  Ex: if [Field]=2 then Case([Field];"1":"Male";"2":"Female") returns "Female"

If [Field]=Japan then Case([Field];"US":"Domestic";"Canada":"North America";"Mexico":"North America";Else:"International") Returns "International"

- **In (reference_value;value1;value2;...)** compares value1, value2, ... to reference_value. As soon as it finds a match, it returns True. If no match is found, it returns False
  Ex: if [Field]=2 then In([Field];"1";"2";"3") returns True

- **Between (reference_value;value1;value2)** returns True if reference_value is equal or higher than value1 and equal or lower than value2. If not, it returns False.
  Ex: if [Field]=2 then Between([Field];"1";"2") returns True

Statistical functions

- **Min(value1;value2;...)** returns the smallest value (numeric values or dates only: empty values are ignored)
  Ex: if [FieldB]=10 and [FieldC]=20 then Min([FieldB];[FieldC])=10

- **MinC(value1;value2;...)** returns the smallest value (characters are accepted, the sort is based on the characters, so in this case 10 is higher than 20)
  Ex: if [FieldB]="ProductA" and [FieldC]="ProductB" then MinC([FieldB];[FieldC])="ProductA"

- **Max(value1;value2;...)** returns the highest value (numeric values or dates only: empty values are ignored)
  Ex: if [FieldB]=10 and [FieldC]=20 then Max ([FieldB];[FieldC])=20

- **MaxC(value1;value2;...)** returns the highest value (characters are accepted, the sort is based on the characters, so in this case 10 is higher than 20)
  Ex: if [FieldB]="ProductA" and [FieldC]="ProductB" then MaxC([FieldB];[FieldC])="ProductB"

- **Count(value1;value2;...)** counts the number of values that are numeric
  Ex: if [FieldB]=1, [FieldC]="X" and [53]=2 then Count([FieldB];[FieldC];[53])=2

- **CountC(value1;value2;...)** counts the number of values that are not blank
  Ex: if [FieldB]=1, [FieldC] is empty(blank) and [53]=2 then CountC([FieldB];[FieldC];[53])=2

- **CountBlank(value1;value2;...)** counts the number of values that are blank
  Ex: if [FieldB]=1, [FieldC] is empty(blank) and [53]=2 then CountBlank([FieldB];[FieldC];[53])=1

- **CountIf(logical_test; value1;value2;...)** counts the number of values that meet the given condition

Ex: if [FieldB]="Yes",[FieldC]="Yes",[53]="No" then
CountIf("="Yes";[FieldB];[FieldC];[53])=2

- **Avg(value1;value2;...)** calculates the arithmetic mean of the values (numeric values only. Blank and characters are ignored)
  Ex: if [FieldB]=10, [FieldC] is empty(blank) and [53]=20 then
  Avg([FieldB];[FieldC];[53])=15

- **Large(k;value1;value2;...)** returns the k-th largest value (numeric values only. Blank and characters are ignored)
  Ex: if [FieldA]=10, [FieldB]=12 and [FieldC]=20 then
  Large(2;[FieldA];[FieldB];[FieldC])=12

- **Small(k;value1;value2;...)** returns the k-th smallest value (numeric values only. Blank and characters are ignored)
  Ex: if [FieldA]=10, [FieldB]=12 and [FieldC]=20 then
  Small(3;[FieldA];[FieldB];[FieldC])=10

AutoNumber function

- **AutoNumber (path_of_file_containing_next_number_to_be_used)** opens the specified path, reads the first line and returns it. If it is a number, it will automatically increment it by one.
  Ex: If you open Notepad, type 1000, save it as a text file called AutoNumber.txt in the Data folder, AutoNumber("AutoNumber.txt") returns 1000 , opens the file, changes the value to 1001 and saves the file.
  NOTE: if the file is not saved in the Data folder, you will need to specify the complete path like C:\MyDocuments\AutoNumber.txt

Date/Time functions

- **Now()** will return the current date.
  Ex: Now() returns "9/14/2004".

- **Time()** will return the current time.
  Ex: Time() returns "9:10:50 PM".

- **Day** will extract the day from a date.
  Ex: if FieldA = "2004/08/31" then Day([FieldA]) returns 31.

- **Weekday** will return the day of the week (Sunday=1, Monday=2, etc.)
  Ex: if FieldA = "2004/08/31" then Weekday([FieldA]) returns 3

- **Weekdayname** will return the name of the day.
  Ex: if FieldA = "2004/08/31" then Weekdayname([FieldA]) returns Tuesday.

- **Month** will extract the month from a date.
  Ex: if FieldA = "2004/08/31" then Month([FieldA]) returns 8.

- **Monthname** will return the name of the month.
  Ex: if FieldA = "2004/08/31" then Monthname([FieldA]) returns August.

- **Year** will extract the year from a date.
  Ex: if FieldA = "2004/08/31" then Year([FieldA]) returns 2004.

- **Hour**, **Minute** and **Second** will extract the hours, minutes and seconds of a date/time.
  Ex: if FieldA = "9:10:50 PM" then Hour([FieldA]) returns 21 (for 9PM).

- **Age(Birthdate;ReferenceDate)** will return the age of a date in years as of the date of the reference date. The reference date is optional. If not mentioned, today's date is used.
  Ex:
  if FieldA = "12/31/1960" and today's date is 12/31/2008 then Age([FieldA]) returns 48.
  if FieldA = "12/31/1960" then Age([FieldA];"12/31/2010") returns 50.

- **DateDiff(interval;date1;date2;firstdayofweek;firstweekofyear)** returns the number of time intervals between two specified dates
  Ex: if [FieldB]="12/12/2003" and [FieldC]="12/15/2003" then
  datediff("d";[FieldB];[FieldC])=3

  You can use the **DateDiff** function to determine how many specified time intervals exist between two dates. For example, you might use **DateDiff** to calculate the number of days between two dates, or the number of weeks between today and the end of the year.

  To calculate the number of days between **date1** and **date2**, you can use either Day of year ("y") or Day ("d"). When **interval** is Weekday ("w"), **DateDiff** returns the number

of weeks between the two dates. If **date1** falls on a Monday, **DateDiff** counts the number of Mondays until **date2**. It counts **date2** but not **date1**. If **interval** is Week ("ww"), however, the **DateDiff** function returns the number of calendar weeks between the two dates. It counts the number of Sundays between **date1** and **date2**. **DateDiff** counts **date2** if it falls on a Sunday; but it doesn't count **date1**, even if it does fall on a Sunday.

If **date1** refers to a later point in time than **date2**, the **DateDiff** function returns a negative number.

Interval		**Firstdayofweek**		**firstweekofyear**	
Setting	**Descr.**	**Value**	**Descr.**	**Value**	**Descr.**
yyyy	Year	1	Sunday	1	Start with week in which Jan1 occurs
q	Quarter	2	Monday		
m	Month	3	Tuesday		
y	Day of year	4	Wednesday	2	Start with the 1st week that has at least 4 days in the new year
d	Day	5	Thursday		
w	Weekday	6	Friday		
wd	Working day	7	Saturday		
ww	Week			3	Start with first full week of the year
h	Hour				
n	Minute				
s	Second				

Firstdayofweek and **firstweekofyear** are optional. If omitted, a value of 1 is assumed.

Working day means Monday To Friday.

- **DateAdd(interval;number;date)** returns a date to which a specified time interval has been added.
  Ex: if [FieldB]="12/12/2003" then DateAdd("d";3;[FieldB])="12/15/2003"

  You can use the **DateAdd** function to add or subtract a specified time interval from a date. For example, you can use **DateAdd** to calculate a date 30 days from today or a time 45 minutes from now.

  To add days to ***date***, you can use Day of Year ("y"), Day ("d"), or Weekday ("w").

  The **DateAdd** function won't return an invalid date. The following example adds one month to January 31:

  ```
  DateAdd("m"; 1; "31-Jan-95")
  ```

  In this case, **DateAdd** returns 28-Feb-95, not 31-Feb-95. If ***date*** is 31-Jan-96, it returns 29-Feb-96 because 1996 is a leap year.

Interval	
Setting	**Descr.**
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week
wd	Working Day
h	Hour
n	Minute
s	Second

Working day means Monday To Friday.

- **DatePart(interval;date;firstdayofweek;firstweekofyear)** returns the specified part of a given date.
  Ex: if [FieldB]="12/12/2003" then DatePart("y";[FieldB])=346 (346th day of the year)

  You can use the **DatePart** function to evaluate a date and return a specific interval of time. For example, you might use **DatePart** to calculate the day of the week or the current hour.

  The ***firstdayofweek*** argument affects calculations that use the "w" and "ww" interval symbols.

Interval		Firstdayofweek		firstweekofyear	
Setting	**Descr.**	**Value**	**Descr.**	**Value**	**Descr.**
yyyy	Year	1	Sunday	1	Start with week in which Jan1 occurs
q	Quarter	2	Monday		
m	Month	3	Tuesday		
y	Day of year	4	Wednesday	2	Start with the 1st week that has at least 4 days in the new year
d	Day	5	Thursday		
w	Weekday	6	Friday		
ww	Week	7	Saturday	3	Start with first full week of the year
h	Hour				
n	Minute				
s	Second				

Firstdayofweek and **firstweekofyear** are optional. If omitted, a value of 1 is assumed.

Format function

The syntax of the format function is the following:

- **Format (expression;format)**
  Ex: if field [FieldB]=2343.3 then Format([FieldB];"###0.00") returns 2343.30

The format argument is a string of characters with one or more of the following characters:

String formatting:

@	A character should appear at this position. If there is no character for this position, a space is inserted. If there are more than one @, they are applied from right to left.
&	Same as @, except that no space is inserted if there is no character at this position.
!	Reverses the order in which @ and & are applied (ie. becomes from left to right).
<	Converts all characters to lowercase.
>	Converts all characters to uppercase.

Ex: Format("AbcD"; ">")= ABCD (similar to the UCase function)
Format("4142345678";"(@@@) @@@-@@@@")=(414) 234-5678
Format("2345678";"(@@@) @@@-@@@@")=( ) 234-5678
Format("2345678";"!(@@@) @@@-@@@@")=(234) 567-8

Number formatting:

	(blank)The digit is returned without formatting
0	A digit is supposed to appear at this position. If there is none, then 0 is displayed. If the format string contains more 0 than the number to be formatted, extra 0s are added in front or at the end.
#	Similar to 0 except that nothing is inserted if no digit is to appear
.	Combined with 0 or #, specifies the number of digits which should appear on each side of the decimal character.
%	Multiplies the number by 100 and adds a % sign.
,	Inserted in series of 0 or #, indicates the thousand separator. A double comma indicates that the number should be divided by 1000.
E-, E+	If the format string contains at least a 0 or #, converts the number to scientific notation.

Ex: Format(2343.3;"0000.00") = 02343.20
Format(2343.3;"$###,###.00") = $2,343.00
Format(45, "+###") = +45

Date and time formatting

D	Displays the date according to your locale's long date format (Windows Control Panel)
d	Displays the date according to your locale's short date format (Windows)
T	Displays the time according to your locale's long time format
Medium Time	Displays the time in 12-hour format using hours and minutes and the AM/PM designator.
t	Displays a time using the 24-hour format

f	Displays the long date and short time according to your locale's format
F	Displays the long date and long time according to your locale's format
g	Displays the short date and short time according to your locale's format
M	Displays the month and the day of a date
R	Formats the date and time as Greenwich Mean Time (GMT)
Y	Formats the date as the year and month
c	Displays the date (as ddddd), the time (as ttttt) if one or both are specified.
d	Displays the day (1 to 31).
dd	Displays the day using 2 digits (01 to 31).
ddd	Displays the day using 3 characters (Sun to Sat).
dddd	Displays the full day (Sunday to Saturday).
ddddd	Displays the date in Short date format (your Windows settings).
dddddd	Displays the date in long time format (your Windows settings).
MM	Displays the month using 2 digits (01 to 12)
MMM	Displays the month using 3 characters (Jan to Dec)
MMMM	Displays the full month (January to December)
T	Displays the time in AM/PM format
y	Displays the day (1 to 366)
yy	Displays the year using 2 digits
yyyy	Displays the full year (1000 to 9999)
h	Displays the hours(0 to 12) 12-hour format
hh	Displays the hours(00 to 12) 12-hour format
H	Displays the hours(0 to 24)
HH	Displays the hours(00 to 24)
m	Displays the minutes (0 to 59)
mm	Displays the minutes (00 to 59)
s	Displays the seconds (0 to 59)
ss	Displays the seconds (00 to 59)
tt	Displays AM or PM

Ex: Format(01/15/2004;"dddd")=Thursday
Format(Now();"T")="12:59:47 PM”

Run function

The syntax of the Run function is the following:

- **Run("path_of_file_to_be_run","parameters")** launches the file if it is an executable or opens the file with the program associated to its extension if any.

  parameters: (optional) parameters to be sent to the program at startup, if any.

MessageBox function

The syntax of the MessageBox function is the following:

- **MessageBox("your_message")** pops up a simple message window which the user can dismiss by clicking OK.

  NOTE: while the message window is displayed, all calculations are halted. The user will need to dismiss the window to continue working with ACT!.

ConfirmationBox function

The syntax of the ConfirmationBox function is the following:

- **ConfirmationBox("your_message")** pops up a message window which the user can dismiss by clicking either Yes or No. If the user clicks Yes, the funtion returns True. If the user clicks No, the function returns False.

  NOTE: while the confirmation window is displayed, all calculations are halted. The user will need to dismiss the window to continue working with ACT!.

PromptBox function

The syntax of the PromptBox function is the following:

- **PromptBox("your_message,allowcancel,title")** pops up a message window with a text box. If the user clicks OK, the function returns the value entered in the text box. If allowcancel is set to 0, the Cancel button is shown and the user needs to enter a value. If allowcancel is set to 1, the Cancel button is displayed and the user may cancel out the prompt window which has the same result as entering no value in the text box, i.e the function returns nothing.

  NOTE: while the prompt window is displayed, all calculations are halted. The user will need to dismiss the window to continue working with ACT!.

Internal Record ID function

The syntax of the InternalRecordID function is simply **InternalRecordID ()** and returns the internal GUID used by ACT! To identify the record.
Its format is xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx.

Here is an example is: 086273c0-12cf-446e-9afe-f09164628d4d

Copy To Clipboard function

The syntax of the CopyToClipboard function is the following:

- **CopyToClipboard("text_to_copy")** will copy the text to the clipboard.

GenerateDocument function

The syntax of the GenerateDocument function is the following:

- **GenerateDocument("template_path")** will launch a merge of the specified template. If the template is in the Act! template folder, simply indicate the name of the file, otherwise indicate the full path.
  Ex: GenerateDocument("03 Presentation - Follow - Up.adt")

Financial functions

- **FV(rate;nper;pmt;pv;type)** returns the future value of an annuity based on periodic, fixed payments and a fixed interest rate.

  The **rate** and **nper** must be calculated using payment periods expressed in the same units. For example, if **rate** is calculated using months, **nper** must also be calculated using months.
  For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

  rate: interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
  nper: total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
  pmt: payment to be made each period. Payments usually contain principal and interest that doesn't change over the life of the annuity.
  pv: (optional) present value (or lump sum) of a series of future payments. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make. If omitted, 0 is assumed.
  type: (optional) when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

- **IPmt(rate, per, nper, pv, fv, type)** returns the interest payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

  The **rate** and **nper** arguments must be calculated using payment periods expressed in the same units. For example, if **rate** is calculated using months, **nper** must also be calculated using months.
  For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

  rate: interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
  per: payment period in the range 1 through **nper.**
  nper: total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
  pv: present value, or value today, of a series of future payments or receipts. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make.
  fv: (optional) future value or cash balance you want after you've made the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
  type: (optional) when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

- **NPer(rate, pmt, pv, fv, type)** returns the number of periods for an annuity based on periodic, fixed payments and a fixed interest rate.

  For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

  rate: interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
  pmt: payment to be made each period. Payments usually contain principal and interest that doesn't change over the life of the annuity.
  pv: present value, or value today, of a series of future payments or receipts. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make.
  fv: (optional) future value or cash balance you want after you've made the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
  type: (optional) when payments are due. Use 0 if payments are due at the end of the

payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

- **Pmt(rate, nper, pv, fv, type)** returns the payment for an annuity based on periodic, fixed payments and a fixed interest rate.

  The ***rate*** and ***nper*** arguments must be calculated using payment periods expressed in the same units. For example, if ***rate*** is calculated using months, ***nper*** must also be calculated using months.
  For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

  rate: interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
  nper: total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
  pv: present value, or value today, of a series of future payments or receipts. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make.
  fv: (optional) future value or cash balance you want after you've made the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
  type: (optional) when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

- **PV(rate, nper, pmt, fv, type)** returns the present value of an annuity based on periodic, fixed payments to be paid in the future and a fixed interest rate.

  The **rate** and **nper** arguments must be calculated using payment periods expressed in the same units. For example, if **rate** is calculated using months, **nper** must also be calculated using months.
  For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

  rate: interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
  nper: total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
  pmt: payment to be made each period. Payments usually contain principal and interest that doesn't change over the life of the annuity.

fv: (optional) future value or cash balance you want after you've made the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
type: (optional) when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

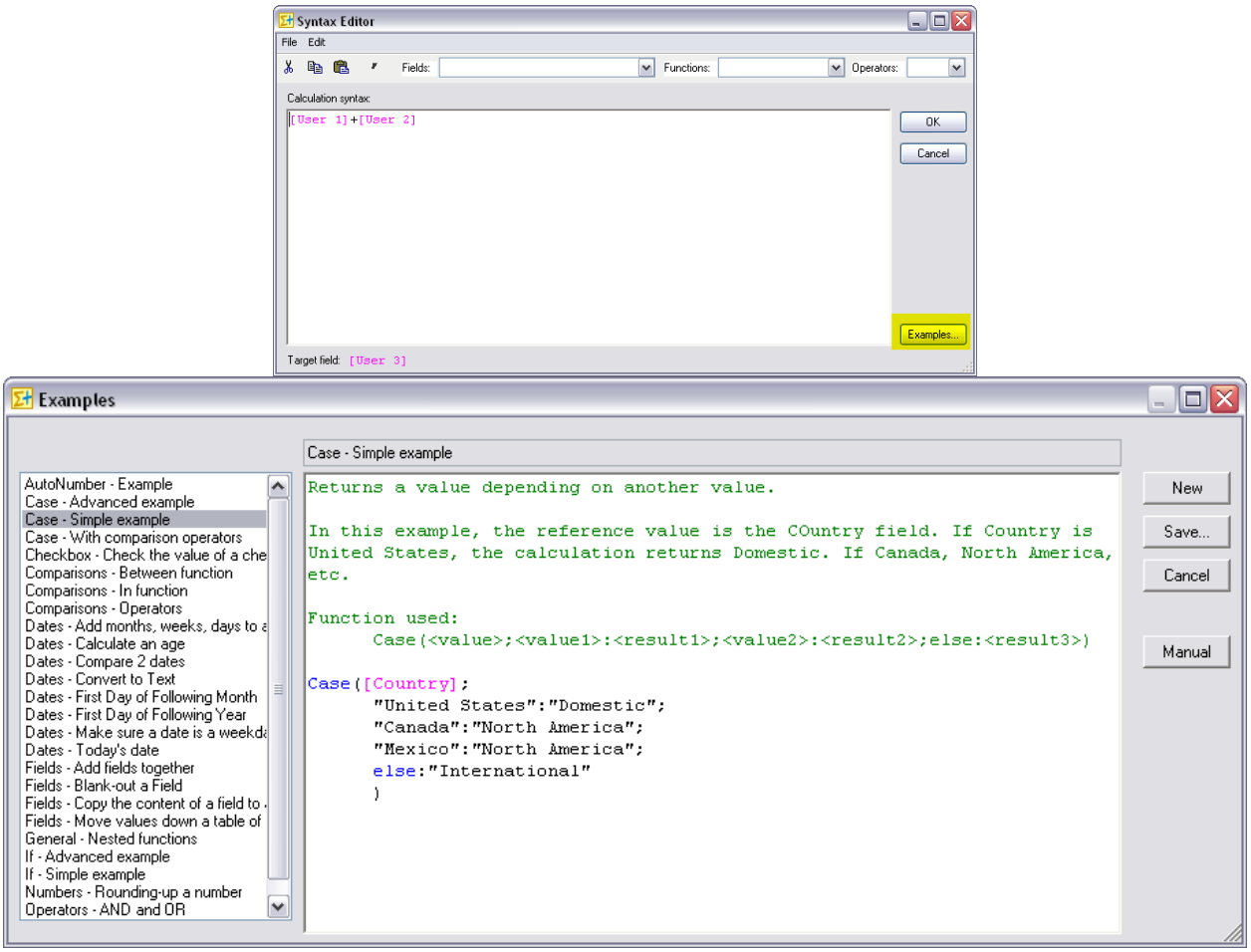
- **Rate(nper, pmt, pv, fv, type, guess)** returns the interest rate per period for an annuity.

  For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers. **Rate** is calculated by iteration. Starting with the value of ***guess***, **Rate** cycles through the calculation until the result is accurate to within 0.00001 percent. If **Rate** can't find a result after 20 tries, it fails. If your guess is 10 percent and **Rate** fails, try a different value for ***guess***.

  nper: total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
  pmt: payment to be made each period. Payments usually contain principal and interest that doesn't change over the life of the annuity.
  pv: present value, or value today, of a series of future payments or receipts. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make.
  fv: (optional) future value or cash balance you want after you've made the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
  type: (optional) when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.
  guess: value you estimate will be returned by **Rate**. If omitted, ***guess*** is 0.1 (10 percent).

Examples of calculations

You can access a number of commented examples by using the File>Examples menu of the Syntax editor, or by clicking the Ex button on the editor's toolbar.



Rev. Nov 20, 2014