TriggerCalc

&

TriggerCalc Pro

v.5.0

for ACT! 2006

Another efficient and affordable ACT! Add-On by

exponenciel

http://www.exponenciel.com

Table of contents

INTRODUCTION 3
Purpose of the add-on 3
How it works 3
Changes from previous versions (ACT! 2000, 5.0, 6.0) 3
Changes from v. 4.0 to v.4.3/5.0 4
Installation procedure 4
Adding icons to other views 4
USER'S MANUAL 6
Creating a calculation 6
Creating a calculation trigger 10
Running a calculation trigger in manual mode 13
Running in manual mode from the icon in your ACT! toolbar 13
Running in manual mode from a button in your layout 13
How to run a calculation trigger in automatic mode using Field Assignments 14
Sharing assignments between databases 15
Sharing your data over a network 15
Synchronizing your calculations, calculation triggers and assignments 16
Errors 16
Using a function inside another function (TriggerCalc Pro only) 16
Yes/No Fields 17
Running triggers for the current lookup or all records 17
Registering TriggerCalc 18
Support 18
REFERENCE GUIDE 19
Operators (Both versions) 19
Basic Operators: 19
Operator precedence 19
Functions (TriggerCalc Pro only) 21
String functions 21
Mathematical functions 22
If function 23
Statistical functions 23
Date/Time functions 25
Format function 28
Run function 29
Financial functions 30
EXAMPLES OF CALCULATIONS 34
Calculating the age of a contact 34
Converting a date into a string 34
Capitalizing the first character of a field 34
Copying the content of a field to another field 34
Formatting a number 34
Blanking out a field 35
Getting the 1st day of the following year 35
Getting the 1st day of the following month 35

INTRODUCTION

Purpose of the add-on

The main objective of TriggerCalc is to allow calculations between fields.

A typical example of use would be that you want the value of FieldB to change when the value of FieldA changes. Using ACT! trigger feature, you could set TriggerCalc to automatically launch when the user exits FieldA, so that FieldB is automatically updated in the event any change was made to FieldA.

The difference between TriggerCalc and TriggerCalc Pro is that the Standard version only support operators whereas the Pro version supports also a number of functions.

How it works

TriggerCalc allows to run calculation triggers. A calculation trigger may contain as many calculations as you want.

Calculation triggers may be run manually or automatically when you exit a field or when you change the value of a field.

TriggerCalc works with Contact fields as well as Company and Group fields. The method for creating calculations is the same: TriggerCalc will automatically detect the active view.

Changes from previous versions (ACT! 2000, 5.0, 6.0)

ACT! 2005 (7.0) has undergone many changes which forced TriggerCalc to be completely redesigned. For instance, since ACT! 2005 does not support command line arguments in triggers, we can not create calculations directly in the trigger field as we used to.

ACT! does not use internal IDs anymore so none of the triggers or trigger files created with previous versions may be used.

Changes from v. 4.0 to v.4.3/5.0

If you owned a version for ACT! 2005 prior to v. 4.3.1, here are two important changes:

- You do not have to add triggers to ACT! fields manually. It is now done through the *File>Assign calculation triggers to fields...* menu (see *Running a calculation trigger in automatic mode* below).
- You do not have to escape semi-colons anymore when using functions inside functions.
- You now have the ability to add buttons to your layouts to run your calculation triggers (see *Running a calculation in manual mode* below).

Installation procedure

Download the program file from our download page and double-click it to start the installation.

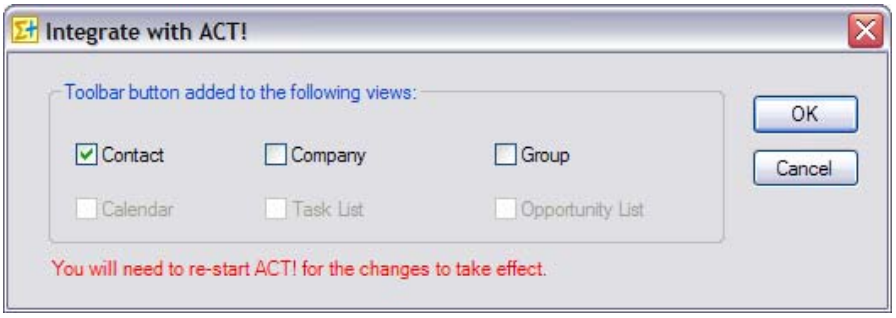
An icon should have been added to your Contact toolbar to launch TriggerCalc directly from within ACT!.

Note that contrarily to versions supporting previous ACT! versions and because of the changes in ACT!, add-ons are now DLL files and not executable files. This means that they are automatically initialized by ACT! when placed in ACT! plugin folder (a subfolder of the main ACT! directory) and cannot be started from the Windows Start menu anymore. This is why you are not given a choice of installation directory during the installation process.

ACT! should automatically add an icon to the Contact Detail window and an item to the Tools menu. Icons and menu items are now the only ways to start your Exponenciel add-on.

Adding icons to other views

To add icons to other toolbars, run TriggerCalc and go to the *Options > Integrate with ACT!* menu. Check the views you want to add icons to and restart ACT!.



When ACT! re-starts, whatever icons you asked to be created should be there to launch TriggerCalc.

NOTE: You may remove the icons from all toolbars by unchecking all checkboxes to avoid your users clicking the icon out of curiosity and messing up with the program's settings. The TriggerCalc item in the Tools menu will remain.

USER'S MANUAL

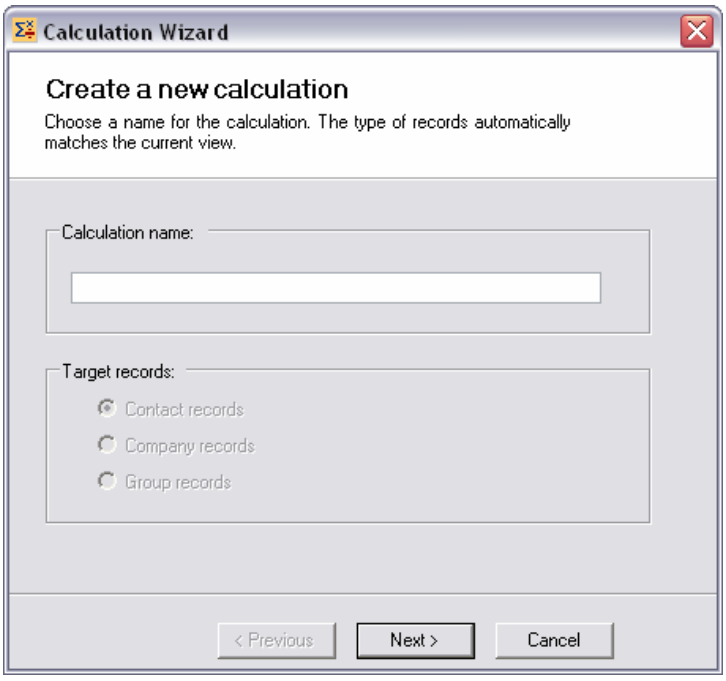
As explained before, TriggerCalc runs calculations triggers. Calculation triggers may contain more than one calculation. To make the creation of calculation triggers and calculations easy, TriggerCalc features a Calculation trigger wizard and a Calculation wizard which you launch from the *File* menu (*New Calculation Trigger* and *New Calculation* menu items).

The first logical step is therefore to create calculations, even though you could create a calculation trigger and then launch the calculation wizard from the last screen of the calculation trigger wizard.

Creating a calculation

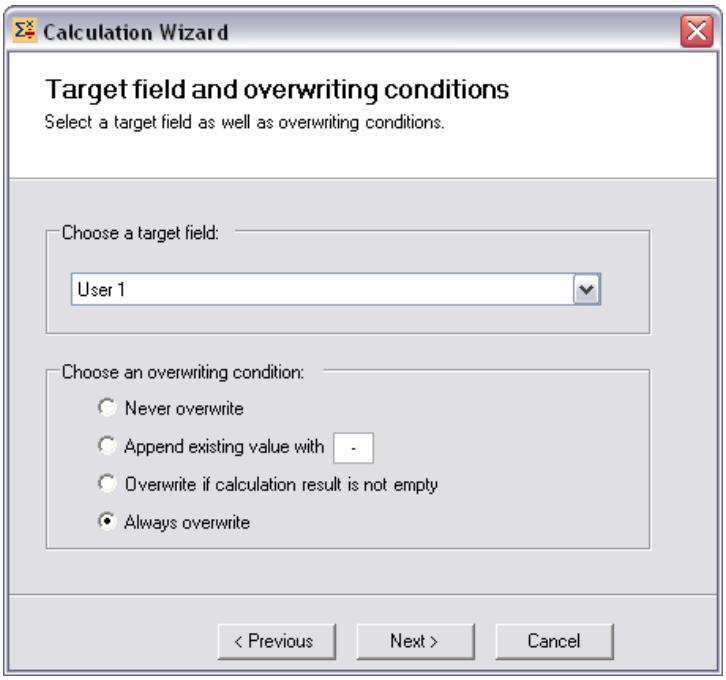
To create a calculation, run TriggerCalc from the contact, company or group view depending on which type of calculation you want to create. Then go to the File menu and choose *New contact calculation...* (which would read *New company calculation...* or *New group calculation...* depending on the view).

Then follow the wizard. In the first screen, you need to choose a name for your calculation.



Note that the Target records type box is dimmed out. It is just indicative of the type of calculation you are creating. The value is automatically set based on the view you were in when you launched TriggerCalc.

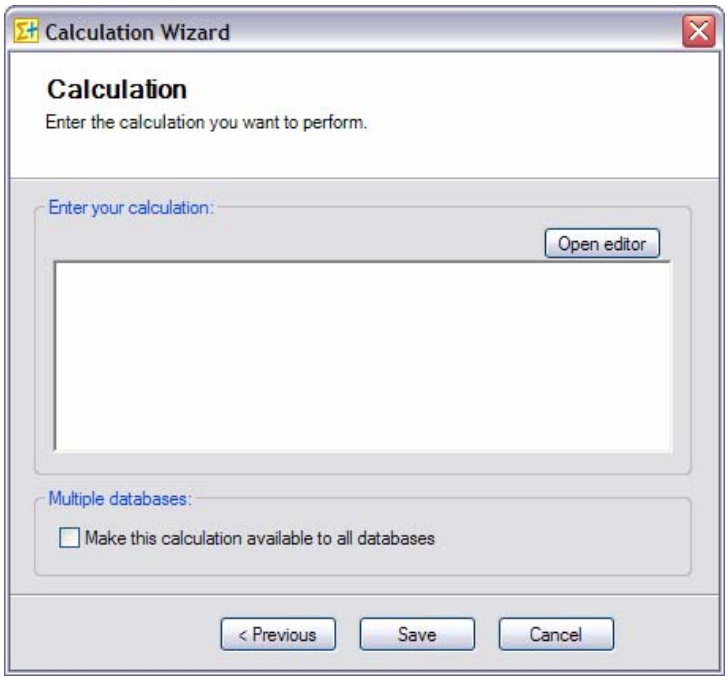
In the second screen, you need to choose a Target field, that is the field in which the result of the calculation will be written, and overwriting conditions.



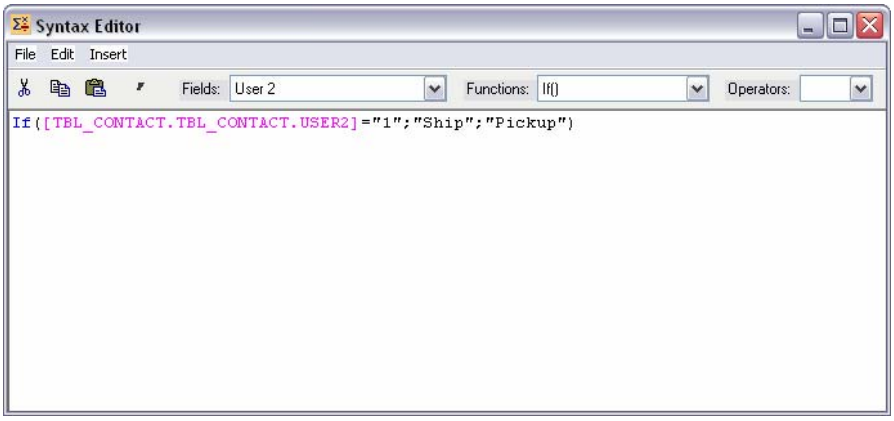
There are 4 overwriting condition options:

- Never overwrite: the result of the calculation will only be written if the target field is empty
- Append existing value with: the result is appended to the end of the data currently in the target field preceded with the append character. If no data is in the field, the result will be written to the field without append character.
- Overwrite if calculation result is not empty: the result will only be written if it is not empty. So if the field contains data and the result is empty, TriggerCalc will not blank out the field.
- Always overwrite: the result will always be written.

In the next field, you may type your calculation. Unless you are copying and pasting a calculation, your best option is to click the *Open editor* button to access TriggerCalc powerful calculation editor.



In the editor, you may use the dropdown lists to insert fields, functions and operators. Color coding is automatic and helps you define your calculation.

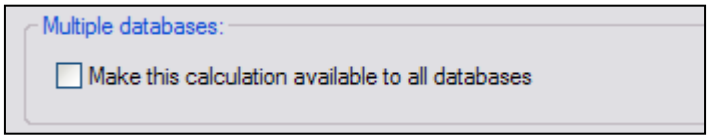


For users of previous versions of TriggerCalc, remember that the target field is set in another screen, so you should type only the calculation you want to perform.

The more complicated your calculation, the more you should be using tabs and comments (the ' button) to make it more readable (see an example in *Using a function inside a function below*).

Once you are done, close the window and say yes to the *Do you want to keep your changes?* prompt. You will be taken back to the last screen of the calculation wizard.

At the bottom of that screen, check the Make this calculation available to all databases if you have more than one database and want to use this calculation in other databases. Use this option with caution as only the basic fields are the same between one database and the other.



If you are satisfied with your calculation, click the *Save* button.

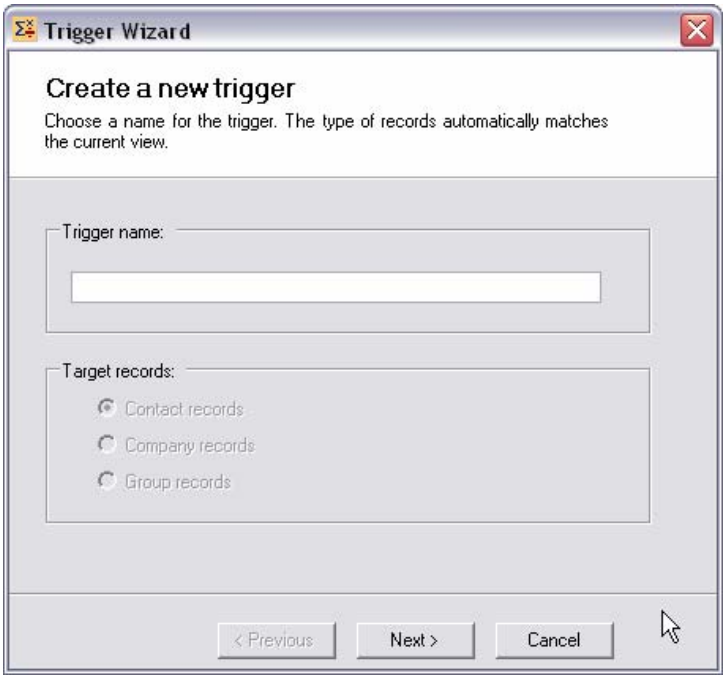
You may define as many calculations as you want.

Creating a calculation trigger

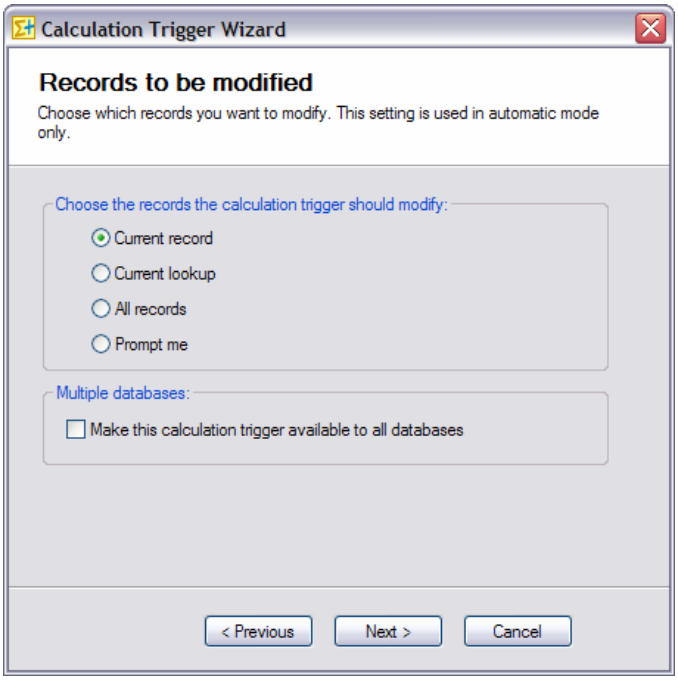
As explained before: in TriggerCalc, you do not run calculations directly, you run calculation triggers. A trigger contains one or more calculations (There are no limit to the number of calculations). Therefore once you created your calculations, you need to create calculation triggers to perform calculations.

The principle of creating a calculation trigger is similar to the one of creating a calculation. To launch the Calculation trigger wizard, go to the *File* menu and choose *New contact calculation trigger*... (or company or group depending on the active view).

In the first screen, you need to name your trigger.



In the second screen, you define the records to be modified.

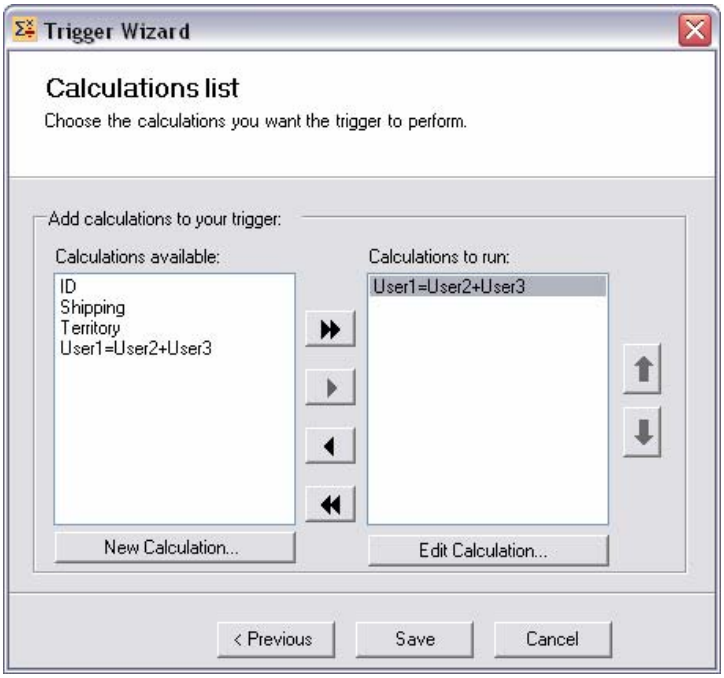


Note that this setting is only used in automatic mode. In manual mode, you will be able to select the proper option manually (see *Running a trigger in manual mode* below).

In the third and last screen, you select the calculations that you want the trigger to run. If you select more than one calculation, the order in which they are run might be important, so make sure they are listed in the right order. They will be run from top to bottom.

If you have multiple databases and want to make this calculation trigger available for all of them, check the box *Make this calculation trigger available to all databases*.

NOTE: If you do check this box, in the next screen, you will be only able to add calculations that are available to all databases.



Finally click *Save*, to save your calculation trigger.

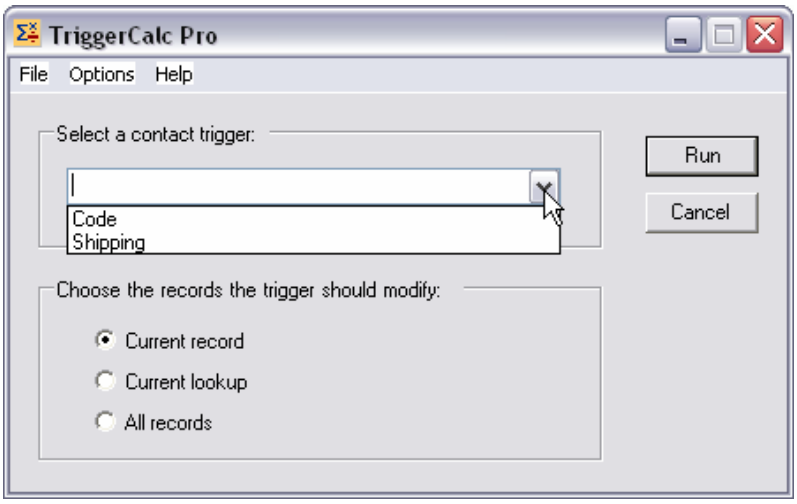
NOTE: As you can see in this last screen, you do have the option to run the Calculation wizard to create New calculations from here. So you could start creating triggers even if no calculations are available and create them at this stage.

Running a calculation trigger in manual mode

You have two options to run a calculation in manual mode.

Running in manual mode from the icon in your ACT! toolbar

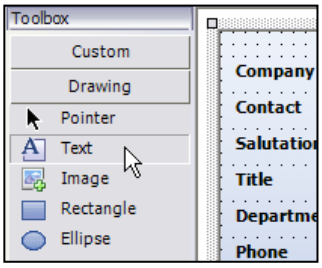
Click the TriggerCalc icon in the ACT! toolbar. Make sure you are in the correct view: if you want to run a contact calculation trigger, you need to be in a contact view. If you want to run a company calculation trigger, then you need to be in a company view. For a group calculation trigger, you need to be in a group view.



Simply select the trigger you want to run, the records it should modify and click Run.

Running in manual mode from a button in your layout

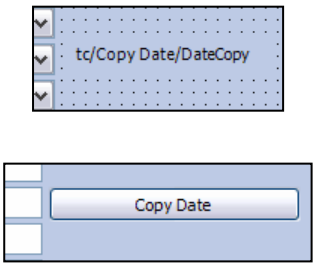
To have TriggerCalc add a button to your layout, in design layout mode, add a label (you need to select the text tool under Drawing) to your layout.



Then type the following text for the label: tc/ButtonCaption/ CalculationTriggerName

- tc stands for TriggerCalc.
- ButtonCaption is the text that will appear on the button.
- CalculationTriggerName is the name of your trigger

Here is an example:



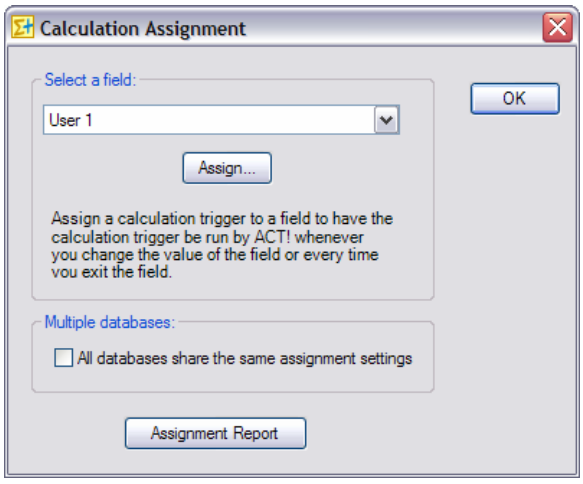
Size and location of the button
The button will be placed at the same location as the label. *It will have the same width as your label, so you may resize it as needed.* The height is automatic and therefore not tied to the height of the label.

When the button is clicked, the specified calculation trigger will be run.

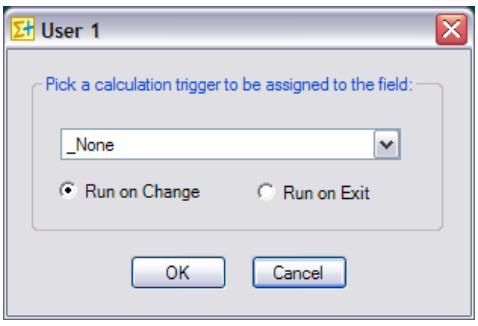
How to run a calculation trigger in automatic mode using Field Assignments

To run a calculation trigger in automatic mode, you need to assign the trigger to one or more fields. Assigning a calculation trigger to a field means that whenever you change the value of the field or whenever you exit the field, the calculation trigger will be run.

To assign a calculation trigger to a field, go to *File > Assign Calculation Trigger to Fields...*



Select the field of your choice and click *Assign...* Then in the next window, pick the calculation trigger to assign to the field.



You have the option to have the trigger be run when the field value changes (*Run on Change*) or when you physically exit the field whether or not the field value was changes (*Run on Exit*).

If you wish to remove a field assignment, simply select *_None* in the list of calculation triggers.

Generally speaking, you would want to assign the calculation trigger to each of the fields involved in the calculations of the trigger.

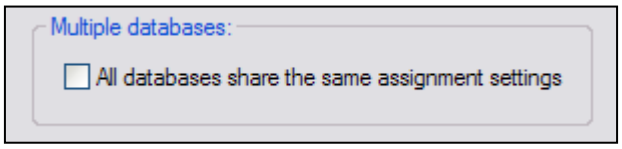
Here is an example to illustrate which fields need to be assigned a calculation trigger.

Let's say you created a calculation trigger that adds User1 to User2 and writes the result in User3. You would want to assign your calculation trigger to User1 and to User2 so that every time User1 and User2 are changed, User3 is automatically re-calculated.
Note: in this scenario, do not associate the trigger to User3. If you do, when the trigger will write the result to User3, it will be automatically re-triggered, entering an endless loop).

The *Assignment Report* button allows to create a text file listing all field assignments.

Sharing assignments between databases

By default, each database has its own field assignments. If your calculation triggers are made available to all your databases, you might want to use the same assignments settings for all your databases. To do this, simply check the *Use the same assignments for all databases* checkbox.



You need to check this box before you assign the calculation triggers. If this box is checked, the assignments are saved in a common settings file. If it is unchecked, they are saved in a database specific settings file.

Sharing your data over a network

If you are in a network environment and are sharing a database among multiple users, you may want to share your calculation settings and field assignments between workstations so that you don't have to recreate them on each PC.

The best way to do this is to do your calculations, assignments setup on the server then to go to the *Options* > *Preferences* menu and note the path to the Data folder. Make sure this folder is shared among all your users.

Then on each workstation, simply go to the *Options* > *Preferences* menu and select the folder on the server for Data folder. From now on, all workstations will read the calculations and assignments settings from your central location. When you make changes to the settings, they are automatically available to all workstations.

Synchronizing your calculations, calculation triggers and assignments

In a synchronization environment, all your satellite databases are different but they share the same field structure. It is therefore possible to share your calculation triggers among them.

Furthermore when you make changes to the calculations on the server, you may want to distribute your changes to the satellite databases.

The best way to do this is to make your calculations and calculation triggers available to all databases (see *Creating a calculation* and *Creating a calculation trigger*). If you automate the calculations, before you assign your calculation triggers, make sure uyou check the box to use the same assignments for all databases (see *Sharing assignments between databases* above).

Your calculation triggers and field assignments are now available to all databases. Simply move the Data folder to a folder that gets sent with ACT! synchronization and point each satellite installation of TriggerCalc to this folder on their local hard drive using the *Options* > *Preferences* menu.

Now whenever you make changes to the central TriggerCalc install, your changes will be propagated to the satellite databases through ACT! synchronization.

Errors

In case the calculation is invalid (like 1 divided by 0 or 50-"A"), TriggerCalc will return #ERR.

Note: *numeric or currency fields in ACT! will not allow accept this message, therefore the field will remain blank or will be blanked out if it previously had a value.*
Therefore you may want to run tests before you convert your fields to numeric or currency, so that you can be made aware of any errors. When you are sure the calculations are properly built, you could then convert your fields to numeric or currency.

Using a function inside another function (TriggerCalc Pro only)

It is possible to use a function in one or more arguments of another function. Starting with v.4.3.1 it is not necessary to escape semi-colons anymore (if you don't understand what this means, just ignore this comment). Make sure you use parenthesis properly though as they are critical to avoid errors.

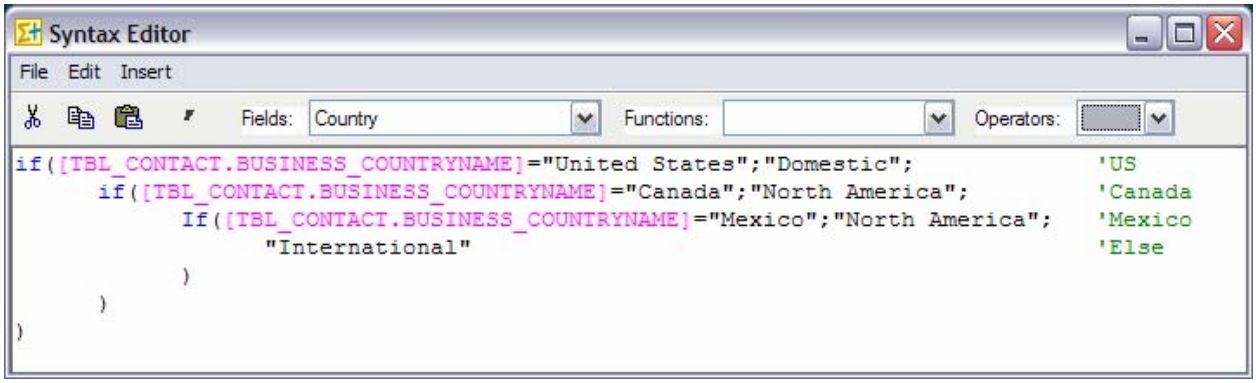
For instance, you may create a function like this one:
If ([FieldA] = "True" ; Avg ([FieldB] ; [FieldC] ; [FieldD]) ; "n.a.")
The Avg function is an argument of the IF function.

Or like this:
Min (if ([FieldB]>=1000; 1000; if([FieldB]>=500; 500; 0)) ; [FieldC])

This example of multi-level calculation could be explained like this:

The target field will be the lowest value of FieldC and a value defined like this:
if FieldB is higher than 1000 then the value is 1000;
If FieldB is higher than 500 but lower than 1000 then the value is 500;
If FieldB is lower than 500, the value is 0

You may want to use comments and indentation to make your code more readable as in:



See how it helps keeping track of opening/closing parentheses.

Yes/No Fields

Yes/No fields return a value of True/False.
If you want to write to a Yes/No field, you could use True/False or a numeric value: 0 is false, anything else is true.

Example of reading a Yes/No Field

```
If([YesNoField];value_when_checked;value_when_not_checked)
```

Example of writing:

```
If([FieldA];True;False )
If( [FieldA];1;0)
If( [FieldA];[NumericField1];[NumericField2])
```

Running triggers for the current lookup or all records

Due to bugs in ACT! 2006, this option is only available for contacts and not for companies nor groups.

Registering TriggerCalc

You may buy licenses from Exponenciel. Registration is based on ACT! user names, so you will have to supply your ACT! username when registering. Once you get your registration code, simply go to the *Help* | *About* menu.

To upgrade from TriggerCalc to TriggerCalc Pro, proceed the same way and overwrite the existing registration code with the new one.

Support

For support, please contact support@exponenciel.com.

REFERENCE GUIDE

Operators (Both versions)

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)

TriggerCalc follows this rule for simple calculations like the one above but for more complicated calculations, **make sure you use parenthesis to help TriggerCalc 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 (TriggerCalc Pro only)

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

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 Harware.

- **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."

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.

Statistical functions

- **Min(value1;value2;...)** returns the smallest value (numeric values only: non-numeric 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 only: non-numeric 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 [FieldB]=10, [FieldC]=12 and [53]=20 then Large(2;[FieldB];[FieldC];[53])=12

- **Small(k,value1;value2;...)** returns the k-th smallest value (numeric values only. Blank and characters are ignored)
  Ex: if [FieldB]=10, [FieldC]=12 and [53]=20 then Small(3;[FieldB];[FieldC];[53])=10

Date/Time functions

- **Now()** will return the current date and time.
  Ex: Now() returns "9/14/2004 9:10:50 PM".

- **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** will return the age of a date in years.
  Ex: if FieldA = "12/31/2003" then Age([FieldA]) returns 41 in 2004 .

- **DaysInMonth** will return the number of days in a specific month.
  Ex: DaysInMonth(2005;02) returns 28.

- **DateSerial** will return a date. It is useful when making calculations (see examples)
  Ex: DateSerial(2005;01;31) returns 01/31/2005 (using your localized date settings).

- **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		
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.

- **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
h	Hour
n	Minute
s	Second

- **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”)** launches the file if it is an executable or opens the file with the program associated to its extension if any.

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

Calculating the age of a contact

If FieldA is the date of birth field:

```
Age([FieldA])
```

Converting a date into a string

If FieldA is the date to convert:

```
Weekdayname(Weekday([FieldA]))& ", "& Monthname(Month[FieldA])& " "& Day([Fiel
dA])&", " & Year([FieldA])
```

Ex: 01/11/2004 is converted into Sunday, January 11, 2004

Capitalizing the first character of a field

If FieldA is a string:

```
UCase(Left([FieldA];1))& Lcase(Mid([FieldA];2))
```

Ex: "Send an Invoice" is converted to "Send an invoice"

Note: to capitalize the first letter of each word, see the Fcase function in the String functions of the standard version.

Copying the content of a field to another field

This is really simple. If FieldA is the field to copy to FieldB, set FieldB as target and use the following calculation:

```
[FieldA]
```

Formatting a number

If FieldA is a number with many digits after the decimal character use the Round function to reduce it to 2 decimals:

```
Round([FieldA];2)
```

Blanking out a field

Simply type a double quote "" in the calculation window and the target field will be reset to a blank value.
(Due to a bug in ACT!, this works only with company and group fields if the field is character-based).

Getting the 1st day of the following year

If FieldA contains a date and you want to get the 1st of the following year, use:
DateSerial(Year([FieldA])+1;01;01)

Getting the 1st day of the following month

If FieldA contains a date and you want to get the 1st of the following month, use:

```
DateAdd("d";1;
        DateSerial(Year([FieldA]); Month([FieldA]);
                DaysInMonth(Year([FieldA]); Month([FieldA]))
                )
        )
```

The idea here is that you add 1 day to the last day of the month. This way you don't have to use an if statement to find out if the date is in December in which case you would have to increment the year as well.