FastExcel V5 SpeedTools



Contents

SpeedTools Overview 7

What's new in SpeedTools Version 5 7
Extended Calculation Methods 7
High-Performance, High-Power Functions 7
Extend your capabilities with over 120 High-Power Functions 7

SpeedTools Array-Handling Functions 9

Array totalling and arithmetic functions 9
Dynamic Array shaping functions 9
Dynamic comparison and merging of ranges and arrays 10
Dynamic sorting functions 10
The Family of LISTDISTINCT Functions 10
High-Performance FILTER.IFS/FILTERX family of functions 11
New Family of AND and OR functions designed for Array Formulas. 12
Efficient dynamic extraction of a Span of Data between 2 values 12

SpeedTools High-Performance Functions 13

High-Performance AVLOOKUP2 family of functions 13
Regular Expression Functions 13
10 New Text-handling Functions 13
New Math and Statistics functions 13

Getting Started with FastExcel SpeedTools 14

Installing FastExcel V5 SpeedTools 14
Automatic Installation for Windows 14
Manual Installation: 14
FastExcel V5 SpeedTools Prerequisites 17
The SpeedTools Ribbon 18
Excel Function Intellisense 19
Excel Function Wizard 19
V5 License Management 20
Buy a license 20
Manage and Release License Activations on other computers 20
Activate/Deactivate License – 20
Deactivate an existing License: 21
Activate a new license 21
SpeedTools Functions by Category 22
SpeedTools Functions Properties 25

Array totalling and arithmetic functions 28

ACCUMULATE Function 28
DIFF Function 30
MOVAVG Function 33
TOTALS & AGGREGATES Functions 37
HTOTALS Function 40
SUMCOLS Function 42
SUMROWS Function 43

AVERAGECOLS Function....45
AVERAGEROWS Function....46
COUNTCOLS Function....47
COUNTROWS Function....48
CORKSCREW Function....49

Array shaping functions 51

UNPIVOT Function....51
SLICES Function....54
SPLICECELLS function....57
SPLICEROWS function....58
SPLICECOLS function....59
REPEAT function....60
RESIZE Function....61
REVERSE.ARRAY Function....62
VSTACK & VSTACKF Functions....63
HSTACK & HSTACKF Function....65
HDR.VSTACK Function....66
HDR.HSTACK Function....68
VECTOR Function....69

Join, Compare and Merge Functions 70

ANTIJOIN MISSMATCH....70
JOIN MATCHES....72
JOIN, ANTIJOIN, MISSMATCHES and MATCHES Examples....73
COMPARE.LISTS Function....76
COMPARE.LISTS Examples....78

Uniques and Distinct array functions 81

LISTDISTINCTS Function....82
LISTDISTINCTS.COUNT Function....84
LISTDISTINCTS.SUM Function....85
LISTDISTINCTS.AVG Function....86
COUNTDISTINCTS Function....87
COUNTDUPES Function....88
LISTDISTINCTS Examples....89

Array Sorting Functions 92

VSORTC – Dynamic text collating Sort of a vertical range or array....94
Case.VSORTC – Case-sensitive dynamic Sort of a vertical range or array....95
VSORTB – Fast Dynamic Sort of a vertical range or array....96
VSORTC.INDEX – Collating Text Index Sort of a vertical range or array....97
Case.VSORTC.INDEX – Collating Text Index Sort of a vertical range or array....98
VSORTB.INDEX – Fast Index Sort of a vertical range or array....99

Logical Functions for Array Formulas 99

OR.ROWS, OR.COLS, OR.CELLS, AND.ROWS, AND.COLS, AND.CELLS, ALL, ANY, NONE....100
Examples of SpeedTools Logical Functions:....102

Array and Range Filtering Functions **105**

The FILTER.IFS Multiple Criteria Function Family 106
FILTER.IFS function....108
FILTER.IFS and ASUMIFS Examples 115
FILTER.SORTED function 118
FILTER.MATCH function 119
ASUMIFS function 120
ACOUNTIFS function 121
FILTER.VISIBLE function 122
Rgx.COUNTIF function 123
Rgx.SUMIF function 124
FILTERX function 125
The SPAN family of functions....130
MATCH.SPAN Function 130
SPAN Function 132
INSPAN Function....133

Lookup and Match Functions **135**

Outstanding Performance 135
Advanced Function 135
Better, Safer Lookup Defaults....136
SpeedTools Lookup Families....136
High-performance exact match Memory Lookups....137
The 24 Advanced Function Lookups 139
MEMLOOKUP Function....141
MEMMATCH Function 144
AVLOOKUP2, AVLOOKUPS2 & AVLOOKUPNTH Functions....147
AVLOOKUP2 Examples....151
Case.AVLOOKUP2, Case.AVLOOKUPS2 & Case.AVLOOKUPNTH Functions 156
AMATCH2, AMATCHES2 & AMATCHNTH functions 160
Case.AMATCH2, Case.AMATCHES2 & Case.AMATCHNTH functions....164
Rgx.AVLOOKUP2, Rgx.AVLOOKUPS2 & Rgx.AVLOOKUPNTH Functions 168
Rgx.Case.AVLOOKUP2, Rgx.Case.AVLOOKUPS2 & Rgx.Case.AVLOOKUPNTH Functions....171
Rgx.AMATCH2, Rgx.AMATCHES2 & Rgx.AMATCHNTH functions 174
Rgx.Case.AMATCH2, Rgx.Case.AMATCHES2 & Rgx.Case.AMATCHNTH functions 177
EVAL2 function: evaluate a string formula 180

Mathematical Functions **181**

VLINTERP2 function 182
LINTERP2D function 184
Calculating Gini Coefficients with GINICOEFF....185
GINICOEFF function 186
IFERRORX Function 187

Reference Functions **189**

PREVIOUS Function....190
SETMEM and GETMEM Functions 191

Information Functions 192

HASFORMULA2 function....193
Calculation Sequence and Counting functions....194
CALCSEQCOUNTREF Function....194
CALCSEQCOUNTSET Function....194
CALCSEQCOUNTVOL function....194
Functions for counting Rows and Columns....195
COUNTROWS2 Function....196
COUNTCONTIGROWS2 Function....197
COUNTUSEDROWS2 Function....199
COUNTCOLS2 Function....200
COUNTCONTIGCOLS2 Function....201
COUNTUSEDCOLS2 Function....203
Examples and comparison of the counting functions....203
Using the Count functions in dynamic range names....204

Text Functions 205

CONCAT.RANGE – concatenate range data....206
PAD.TEXT function....207
REVERSE.TEXT Function....208
SPLIT.TEXT Function....209
GROUPS Function....210
Rgx.LEN function....214
Rgx.SUBSTITUTE function....215
Rgx.MID function....216
COMPARE function....217
ISLIKE2 array function for pattern-matching strings....218
Rgx.ISLIKE function....219

Formatting Dynamic Arrays 221

Edit DA Formats....221
Reformat This DA....223
Show DA Formats....223
Edit DA Format....223
GoTo DA....223
Delete DA Format....223

SpeedTools What-If Tables Manager 224

FastExcel What If Calculation....224
The Manage What-Ifs Userform....224
Setup/Change a What-If Table....225
What-IF Data Table examples....227
The What If Manager Ribbon....228

SpeedTools Calculation Manager 229

Calculation Manager Options and Settings....230
Excel Calculation Settings....231
Excel Current Calculation Mode....232
FastExcel Active Workbook Mode....233
Excel Set Book Modes....234

Excel Calculation Settings: Initial Calculation Mode 235
Excel Calculation Buttons 237
Workbook Calculation Settings 239
Workbook Calculation Settings: Mixed Mode 239
Workbook Calculation Settings: Workbook Options 241
FastExcel Settings 244
Getting Consistent Results from FastExcel V5 Timing 246

Calling SpeedTools functions from VBA **247**

Using FastExcel calculation methods from VBA **248**

MICROTIMER function 249
MILLITIMER function 249
STRCOLID function 250

SpeedTools Overview

What's new in SpeedTools Version 5

New and Extended Functions

- The IFS family of functions (ASUMIFS, FILTER.IFS etc) now accept both Ranges and Arrays.
- The FILTERX function combines the efficiency and power of the IFS family with the array filtering methods of FILTER.
- MATCH.SPAN, SPAN and INSPAN functions provide simple and efficient ways of extracting an inclusive block of data between two values.
- The IFS family and FILTERX allow 3 different types of list conditions (Include, Exclude and All-Of)
- JOIN and ANTIJOIN functions have been added as alias's of MATCHES and MISSMATCHES
- The GROUPS function now returns arrays of results from arrays and ranges containing text

What-If Tables Manager

What If Manager enables you to speed up and Manage Excel What-If Data Tables.

- Slow Excel Data Tables calculate much faster when converted to FastExcel What-If Tables
- Convert Excel What-If Data Tables to FastExcel What-If Data Tables
- Convert FastExcel What-If Data Tables back to Excel What-If Data Tables

Updated Calculation Manager

The SpeedTools Calculation Manager has been update to include updates to Excel's calculation engine, including Format Stale Values, Partial Calculation and Compatibility version.

Extended Calculation Methods

SpeedTools Extended Calculation methods give you much finer control over Excel calculation.
This finer control allows you to minimize calculation time with slow-calculating workbooks.
When multiple workbooks are open:

- Only calculate the active workbook
- Use different calculation methods (Manual, Automatic for the open workbooks

Within a workbook control assign different calculation methods to each worksheet (Mixed Mode Sheets)
Control the calculation method to use at workbook open time.
Automatically restore each workbook's calculation mode after it is opened.

High-Performance, High-Power Functions

Eliminate LOOKUP, SUMPRODUCT and Array Formula Bottlenecks

Two major Excel calculation bottlenecks are Lookup functions and multiple condition FILTER formulas or their SUMPRODUCT equivalents.
FastExcel SpeedTools now has the solution to many of these bottlenecks with the AVLOOKUP2 and FILTER.IFS families of high-performance functions.

Extend your capabilities with over 120 High-Power Functions

- **Exploit the power of Dynamic Arrays with 50 Array-Handling functions**
  - Dynamically floating totals
  - Accumulate, Difference and Moving Average array calculations
  - Stack Arrays
  - Unpivot
  - Merge (Join) dynamic arrays and find miss-matches (Anti-Join)
  - Resize & Reshape Arrays

- Use **MATCH.SPAN, SPAN** and **INSPAN** to efficiently get the span of values between 2 values
- Use **MEMMATCH, MEMLOOKUP** and a family of **24 advanced function Lookup functions** for
  - Faster exact match Lookups with 3 Memory options
  - Multi-threaded C++ XLL for faster calculation Excel calculation.
  - Fast Exact match option with sorted data
  - Multiple Lookup answers
    - Find the Nth, first, last, all Lookup answers
  - Case-sensitive Lookup option
  - Wild-card and Regular Expression Lookups
  - Multiple Lookup Values
  - Multiple Lookup Columns
  - Multiple Answer Columns
  - Use column labels rather than column numbers
  - Built-in error handling
- Use the **LISTDISTINCTS** family of functions to
  - Work with distinct rows or distinct cells within a multi-column range
  - Find the total number of distincts and duplicates
  - List the distincts
  - Count the number of occurrences for each distinct
  - Sum or average corresponding values for each distinct row
  - Find distincts using multiple criteria using LISTDISTINCTS(FILTER.IFS)
  - List output can be sorted.
  - Case-Sensitive or not
  - Options to ignore cells containing Errors, Blanks or zeros.
- Use **ASUMIFS** and **ACOUNTIFS** for fast and powerful multiple-criteria summing and counting.
- Use **FILTERX, FILTER.IFS, FILTER.SORT** and **FILTER.MATCH** to add fast and powerful multiple criteria capability to many Excel functions such as MAX, MEDIAN etc.
- 10 new Text Functions
  - Use Regular Expressions to find (**Rgx.FIND , Rgx.LEN**), substitute (**Rgx.SUBSTITUTE**) or extract (**Rgx.MID**) within text-strings
  - Concatenate Ranges (**CONCAT.RANGE**)
  - Split (**SPLIT.TEXT**), Pad (**PAD.TEXT**) and Reverse (**REVERSE.TEXT**) text-strings.
  - Extract groups of characters (**GROUPS**) from within text-strings
  - **COMPARE** to compare values in the same sequence as Excel's SORT
- Specialist high-performance functions **VLINTERP2, LINTERP2D** and **GINICOEFF**
- **EVAL2** to evaluate string expressions as formulas or array formulas
- **ISLIKE** and **Rgx.ISLIKE** allow extended wild-card and Regular Expression pattern matching in ordinary and array formulas.
- 6 counting functions specially designed to extend the power of dynamic range names.
- Calculation sequence and counting functions to help you understand Excel calculation quirks.

SpeedTools Array-Handling Functions

These array-handling functions are designed to work with Dynamic Arrays.

Array totalling and arithmetic functions

Functions to provide totals (static or floating below or alongside dynamic arrays), and series calculations for dynamic arrays.

- TOTALS - Floating/fixed totals of the columns in an array or range
- HTOTALS - Floating/fixed totals of the rows in an array or range
- SUMCOLS - Sum columns in an array or range
- SUMROWS - Sum rows in an array or range
- AVERAGECOLS - Average columns in an array or range
- AVERAGEROWS - Average rows in an array or range
- COUNTCOLS - Count the values in columns in an array or range
- COUNTROWS - Count the values in rows in an array or range
- CORKSCREW - Simple arithmetic roll-forward for dynamic arrays
- ACCUMULATE - Carry-forward by row or column in a series
- DIFF - Calculate the difference between successive items in a series
- MOVAVG - Calculate one of 6 types of moving average for a series

Dynamic Array shaping functions

Transform, resize, modify and stack dynamic arrays

- UNPIVOT - Transforms table columns from a range, array or table into rows.
- SLICES - Create a new array from one or more slices of an array or range
- SPLICECELLS - Replace up to 5 cells in an array
- SPLICEROWS - Insert and/or remove rows from an array or range.
- SPLICECOLS - Insert and/or remove columns from an array or range.
- REPEAT - Copy a block of cells on or more times vertically and/or horizontally.
- RESIZE - Resize the rows/columns in an array or spill reference
- VSTACK,VSTACKF- Appends Ranges/Arrays row-wise below one another
- HSTACK,HSTACKF- Appends Ranges/Arrays column-wise alongside each other
- HDR.VSTACK - Appends columns with matching headers from Ranges/Arrays below one another
- HDR.HSTACK - Appends rows with matching headers from Ranges/Arrays alongside one another
- REVERSE.ARRAY - Reverse the rows, columns or both in an array.
- VECTOR - Outputs a sequence of numbers as a column or row vector

Dynamic comparison and merging of ranges and arrays

Efficient ways of dynamically merging and reconciling lists of data.

- JOIN & MATCHES - Join 2 arrays or ranges together on matching key columns
- ANTIJOIN & MISSMATCHES - Find the mismatched rows using key columns (Anti-Join)
- COMPARE.LISTS - Very efficient comparison of two arrays or ranges

Dynamic sorting functions

If you want to dynamically sort the results of a calculation or the output of an array function such as LISTDISTINCTS you can use the 6 dynamic sorting functions with case-sensitive and index sort options. 4 of these functions provide the same sort sequence as the Excel SORT command and can be used to prepare the input for all the LOOKUP functions.

- VSORTC - Dynamic multi-column sort of a vertical range/array
- VSORTC.INDEX - Like VSORTC but returns a single column of indexes to the data
- VSORTB - Like VSORTC but uses a fast binary comparison
- VSORTB.INDEX - Like VSORTC.INDEX but uses a fast binary comparison
- Case.VSORTC - Case-sensitive VSORTC
- Case.VSORTC.INDEX - Case-sensitive VSORTC.INDEX

The Family of LISTDISTINCT Functions

This new family gives you an easy, powerful and efficient way of dynamically working with data containing duplicates.

Work with multiple columns

The ByRows option allows you to look for distinct rows with multiple columns

Case-sensitive Option

You can choose whether to ignore upper-lower case or not.

The LISTDISTINCTS family can return arrays of the distinct items. To simplify dynamically using the results of these functions you can choose to return, 0,"" or #N/A for unused cells.

Counts, Sums and Averages for the list of Unique Items

LISTDISTINCT.COUNT, LISTDISTINCT.SUM, LISTDISTINCT.AVG, COUNTDISTINCTS and COUNTDUPES give you an easy way of dynamically counting the number of occurrences of each distinct item/row, or of summing or averaging a range of values for each distinct row.

- LISTDISTINCTS - Create an array or range list of the distinct cells or rows.
- LISTDISTINCTS.SUM - Produces a list of the distinct items and lists the sums of their corresponding values
- LISTDISTINCTS.COUNT - Produces a list of the distinct items and their count
- LISTDISTINCTS.AVG - Produces a list of the distinct items and their average

High-Performance FILTER.IFS/FILTERX family of functions

The FILTERX, FILTER.IFS, FILTER.MATCH, FILTER.SORTED, ASUMIFS and ACOUNTIFS functions provide you with a high-performance, high-function solution to multiple criteria problems that previously required slow SUMPRODUCT or FILTER formulas.

Now fully supports both Ranges and Arrays

Outstanding performance improvements with sorted data.

The FILTER.IFS family of functions has been implemented using ultra-efficient binary search algorithms that give stunning performance on sorted data.

Efficient performance with clustered or sparse results from unsorted data.

Special care has been taken to minimize search time for unsorted data by exploiting clustered data and subsets of results. For many cases this gives substantial performance improvements on unsorted data.

For worst-case data (50% of data is results randomly selected from unsorted input data) performance will be comparable to or slightly worse than SUMPRODUCT.

Efficient handling of full-column criteria.

Processing is restricted to the used range rather than explicitly checking every row in the column.

Give Multiple-criteria ability to other Excel Functions and UDFs

You can embed the FILTER.IFS functions inside virtually any built-in or UDF function that can handle a Range as input. This extends multi-criteria function to an incredible range of functions, for example: LISTDISTINCTS, COUNTDISTINCTS, MAX, MIN, SUM, COUNT, COUNTA, AVERAGE, MEDIAN, MODE, LARGE, INDEX, VAR, RANK ….

Built-in OR to eliminate double-counting.

Sets of Criteria can be separated by #OR#. The results from multiple sets of criteria are OR together but never double-counted as can happen when using SUMPRODUCT(…) + SUMPRODUCT(…)

Use Lists for alternate criteria

Where you have multiple possible criteria for a single column (FRUIT can be Apples, Oranges or Pears) you can use either a reference to a Range or array containing the alternatives. You can even use different conditions for each element in the List (MonthNumber=2 or >8).
Lists can be inclusion (include any of) or exclusion lists (exclude all of) or all-of (match every condition) lists.

Wild-Card and Regular Expression pattern-matching criteria

You can use wild-card and regular expression patterns for string criteria. The patterns can look for combinations of characters and numbers using powerful pattern-matching function.

Wide variety of Criteria Operators

In addition to the usual criteria operators =, <, <=, >, >=, <> you can use

~ (Like), ~~ (Regular Expression), True, False,

And Data Type filters #ERR, #TXT, #N, #BOOL, #EMPTY, #ZLS, #TYPE, #BLANK

Prefixing any of the criteria operators with ¬ makes the criteria an exclusion criterion rather than an inclusion criterion.

Use Column Labels or Column Numbers for Criteria

Using Column labels from the first row of the data makes the FILTER functions easier to read and you don’t have to remember to change the column numbers in all your formulas when you add columns.

Create virtual calculated criteria columns.

Unlike SUMIF and COUNTIF you can use expressions containing Excel functions and formulas to create virtual calculated columns for your criteria columns.

New Family of AND and OR functions designed for Array Formulas.

Excel's standard OR and AND functions do not generally work well in array formulas because they only return a single True or False rather than evaluating each row or column in the array in turn to return an array of True/False..

The SpeedTools functions OR.ROWS, OR.COLS, OR.CELLS, AND.ROWS, AND.COLS, and AND.CELLS are designed to simplify the use of logical functions in array and FILTER formulas and can be nested to build complex logical array expressions.

- OR.COLS - Logical OR by columns
- OR.CELLS - Logical OR by cell
- OR.ROWS - Logical OR by rows
- AND.COLS - Logical AND by columns
- AND.CELLS - Logical AND by cell
- AND.ROWS - Logical AND by rows

Efficient dynamic extraction of a Span of Data between 2 values

MATCH.SPAN Returns the first and last row or column numbers for a span defined by a From_Value and To-Value in a single column or row of an array or range. Uses efficient Binary Search on sorted data.

SPAN Extracts a span of rows and columns defined by a pair of row numbers and a pair of column numbers from a range/array. Used in conjunction with MATCH.SPAN.
Use **INSPAN** in FILTER. FILTERX and FILTER.IFS formulas to return an array of Trues & Falses for a span defined by a From-Value and To-Value in a single column or row of an array or range

SpeedTools High-Performance Functions

High-Performance AVLOOKUP2 family of functions

The AVLOOKUP2 and AMATCH2 functions have been re-written as multi-threaded C++ XLL functions to significantly improve performance.

The AVLOOKUP2 family can use 4 different kinds of Lookup Memory so that you can choose the optimum solution for your scenario. Lookup Memory is now stored with the workbook so that it is immediately active when you reopen a workbook.

Options are now available for all combinations of:

- Finding the Nth of multiple matches
- Case-sensitive Lookup
- Regular Expression Lookup, supporting ECMASCRIPT/PERL regular expression syntax.

Simple Replacement of VLOOKUP, HLOOKUP and MATCH

The new MEMLOOKUP and MEMMATCH provide a very simple way of speeding up exact match lookups by replacing VLOOKUP, HLOOKUP and MATCH.

MEMLOOKUP and MEMMATCH share the same Memory components as the AVLOOKUP2 family but do not have the many added features of the Advanced VLOOKUP family.

Regular Expression Functions

In addition to the Regular Expression Lookup functions FastExcel SpeedTools has Regular Expression functions for summing, counting and manipulating text:

- Rgx.COUNTIF – count the number of cells that match a regular expression pattern
- Rgx.SUMIF – sum cells whose corresponding cells match a regular expression pattern
- Rgx.IsLike – returns True if the cell matches a regular expression pattern
- Rgx.FIND – finds the position within a string that matches a regular expression pattern
- Rgx.LEN – returns the length of the substring within a string that that matches a regular expression pattern
- Rgx.MID – extracts text that matches a regular expression pattern
- Rgx.SUBSTITUTE – replaces substring(s) that match a regular expression pattern with new text

These functions are all multi-threaded and have been built using the Boost Regular Expressions Library.

10 New Text-handling Functions

Use Regular Expressions to find and manipulate text strings.

Split, Reverse, Pad and extract groups of characters from text strings

Concatenate ranges using delimiters.

New Math and Statistics functions

Use GINICOEFF for efficient calculation of Gini Coefficients (a widely used measure of inequality).
Use the power of Regular Expressions in COUNTIF and SUMIF

Extended vertical and 2-dimensional linear interpolation

Getting Started with FastExcel SpeedTools

Installing FastExcel V5 SpeedTools

FastExcel V5 SpeedTools Installers can either be installed manually from a downloaded zip folder, or more automatically from a downloaded Installer package.

Automatic Installation for Windows

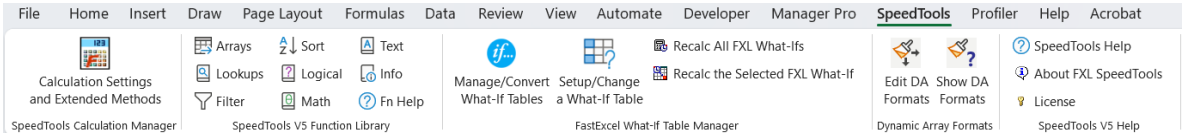
Automatic Installation requires administrative privileges.

1. Extract the installer .exe file from the downloaded zip

2. Double-click the extracted installer .exe file and follow the instructions to create a FastExcel V5 folder containing a subfolder for FXLV5 SpeedTools. The default directory is called FastExcel V5 and is located in your Program Files (x86) directory.

Help files (.CHM) and a PDF version of the User guide will also be installed in these folders.

After successful installation FastExcel V5 will automatically be started when you start Excel, and you will find FastExcel V5 tabs for each installed product on the main ribbon.



If the ribbon does not show any FastExcel V5 tabs, or the installation was done for you by another user with administrative privileges, you may have to use the Manual Install instructions below:

Manual Installation:

- Does not require Administrative Privileges.
- Requires use of Excel's Addin Manager
- Requires Manual Uninstall

Manually Installing FastExcel SpeedTools V5 does not generally require administrative privileges (but some administrative group policies and anti-virus software may prevent downloading and installing Excel addins).

1. Create the FastExcelV5 folder

Extract the FastExcelV5 folder from the downloaded zip file to your chosen location

The FastExcelV5 folder contains up to 3 sub-folders, depending on which Manual Installer you chose:

- FastExcel V5 Profiler
- FastExcel V5 Manager Pro
- FastExcel V5 SpeedTools

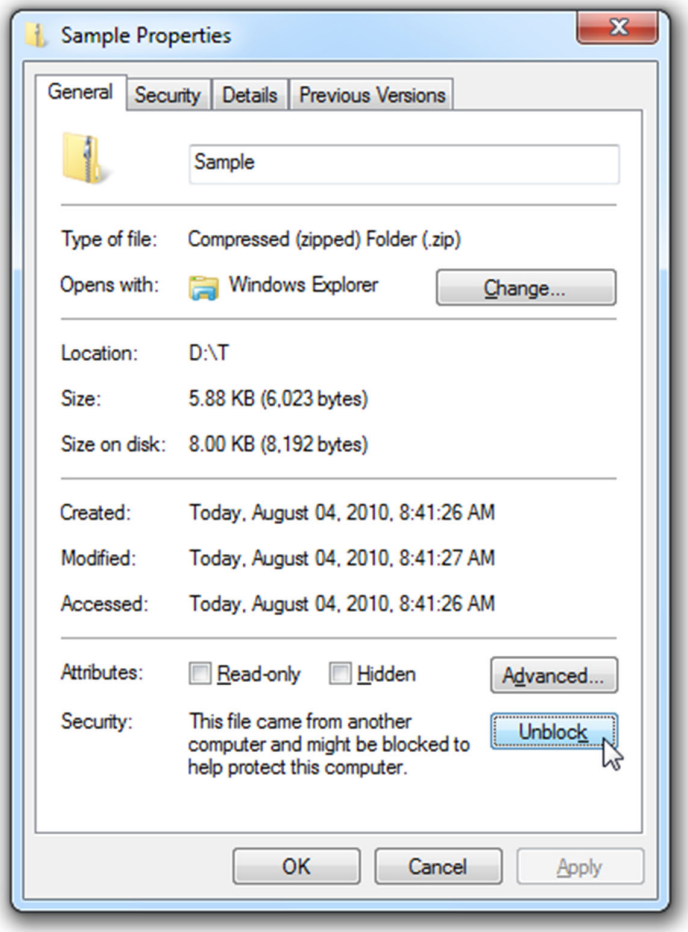
Each of these sub-folders contains an .XLAM file you need to load into Excel using Excel's Addin Manager.

2. Unblock all the XLAM, XLL and DLL files

Windows often blocks downloaded files.

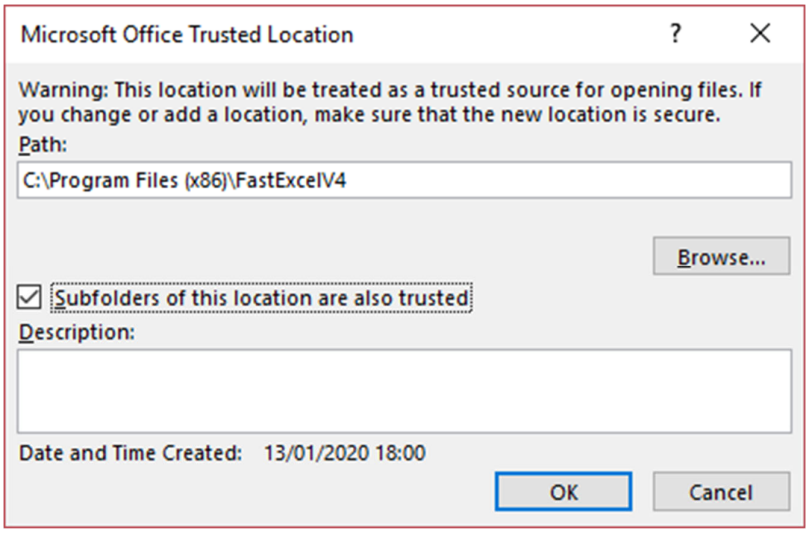
To unblock a file:

- The files you may need to unblock are:
  - FastExcelV5/FastExcel V5 SpeedTools/FxlV5SpeedTools.xlam
  - FastExcelV5/FastExcel V5 SpeedTools/FastExcelV5_32.xll
  - FastExcelV5/FastExcel V5 SpeedTools/FastExcelV5_64.xll
  - FastExcelV5/FastExcel V5 SpeedTools/ExcelDNA.IntelliSense.xll
  - FastExcelV5/FastExcel V5 SpeedTools/ExcelDNA.IntelliSense64.xll
  - FastExcelV5/FastExcel V5 SpeedTools/QlmCLRHost_x64.dll
  - FastExcelV5/FastExcel V5 SpeedTools/QlmCLRHost_x86.dll
  - FastExcelV5/FastExcel V5 SpeedTools/QlmLicenseLib.dll
- Browse to each file, Select and Right-click it and then select Properties
- Click the Unblock button



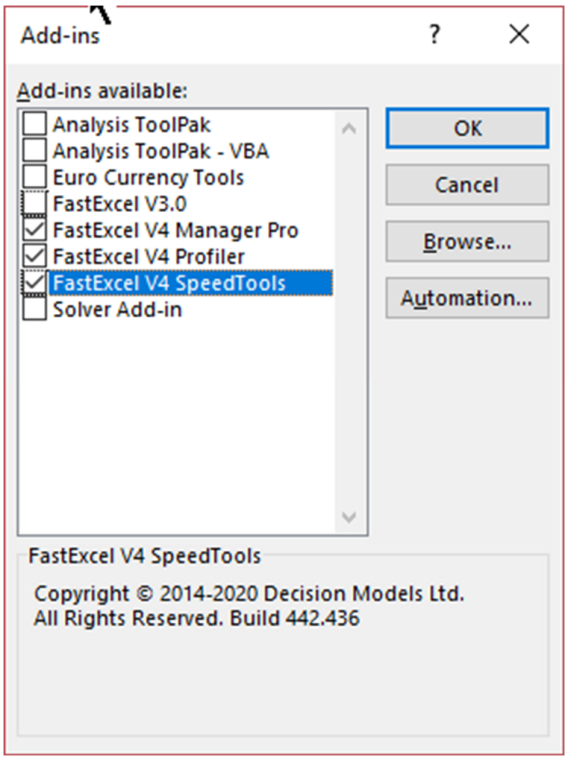
3. Make the FastExcelV5 folder a Trusted Location

Start Excel and use File->Options->Trust Center->Trust Center Settings->Trusted Locations->Add new location->Browse to your FastExcelV5 folder->Check Subfolders of this Location are also trusted->OK



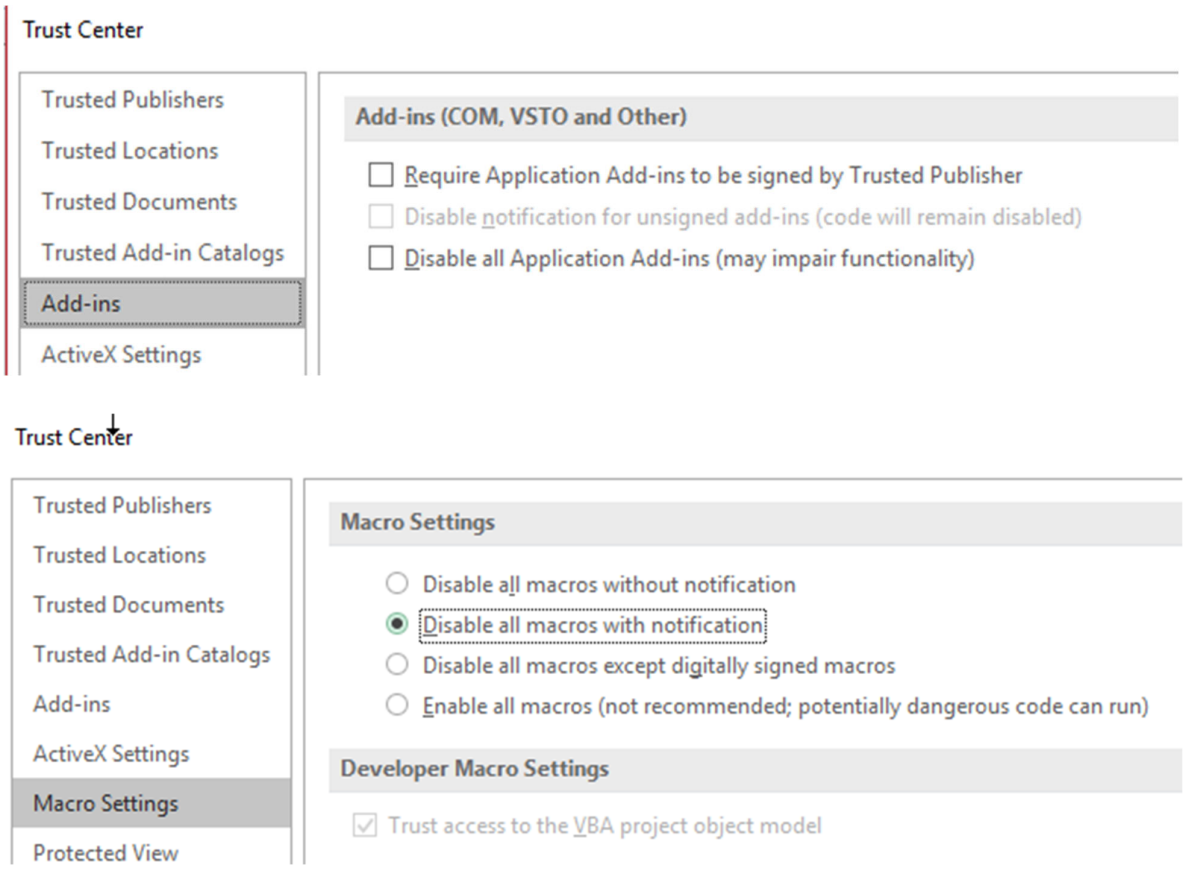
4. Use Excel's Addin Manager to install the FastExcel V5 addin
For Excel 365 Click File->Excel Options->Addins->Manage Excel Addins->Go…

- Press Browse and locate the FastExcelV5 folder.
- In the FastExcel V5 SpeedTools sub-folder select the **fxlV5SpeedTools.xlam** file and click OK to return to the Addins form.
- If asked "Do you want to copy this Addin to the Addins folder?" reply NO.
- The Excel Addins form should now show the FastExcel V5 addins with a checkmark. Click OK to finish.



5. Recommended Trust Center Settings

The recommended settings for addins and macros are:

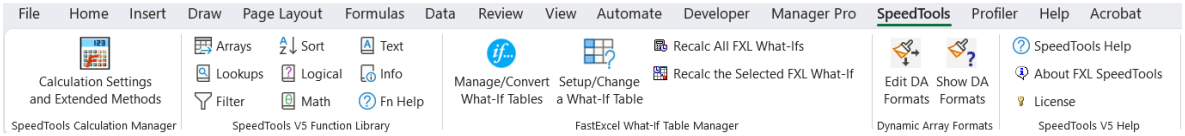


FastExcel V5 SpeedTools Prerequisites

FastExcel V5 requires the following.

- Any Excel version from Office 365.
- 32 or 64 bit Excel
- 32 or 64 bit Windows
- .NET 4

The SpeedTools Ribbon



SpeedTools Calculation Manager gives you:

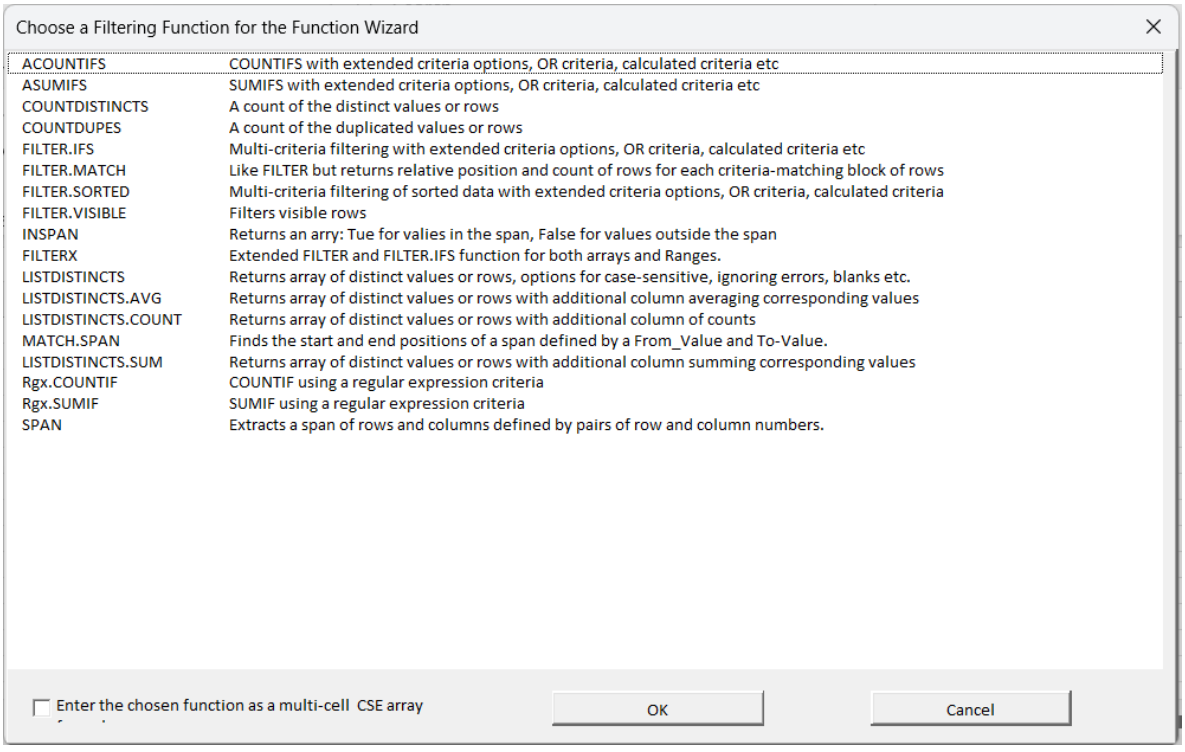
- Extended Calculation Methods to eliminate unnecessary calculations.
    - Active workbook mode – only calculate the active workbook
    - Control which worksheets get calculated (MixMode worksheets)
    - Control Initial calculation mode at Excel startup
    - Restore calculation mode after workbook open
    - Calculate MixMode sheets at Workbook open
- Full Control of all Excel's calculation settings

What-If Table Manager converts slow Excel What-If tables to/from FastExcel What-If Tables**.**

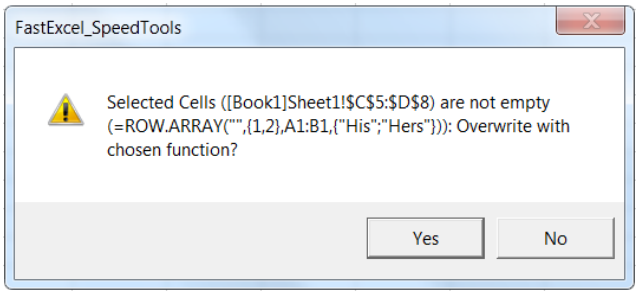
Dynamic Array Formats automatically reformat to match resized dynamic arrays.

The SpeedTools functions are available from the SpeedTools Function Library group on the FXL V5 SpeedTools Tab, as well as being integrated into the Excel Function Wizard and function categories.

Clicking a function group button (for example Filtering) shows you a list of the available functions in that function group, together with a short description.



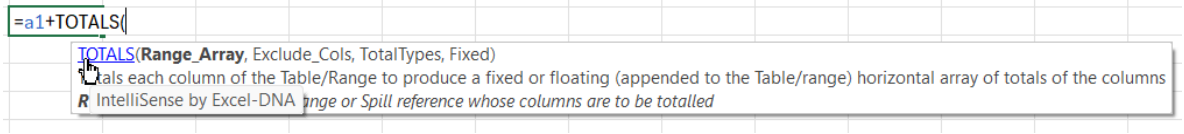
You can select a function and choose whether to enter it as a multi-cell CSE array formula or not. Then clicking OK will enter the function into the selected cells and launch the Excel Function Wizard.
If the selected cells already contain a formula or data you will be asked if you want to overwrite the existing information.



Excel Function Intellisense

SpeedTools functions support Excel IntelliSense.

As you start to enter the name of the function a tooltip pops up showing a short description of the function and its parameters. Clicking the Name of the function shows SpeedTools help for the function.



If you want further help you can invoke the Excel Function Wizard using Control-A.

Excel Function Wizard

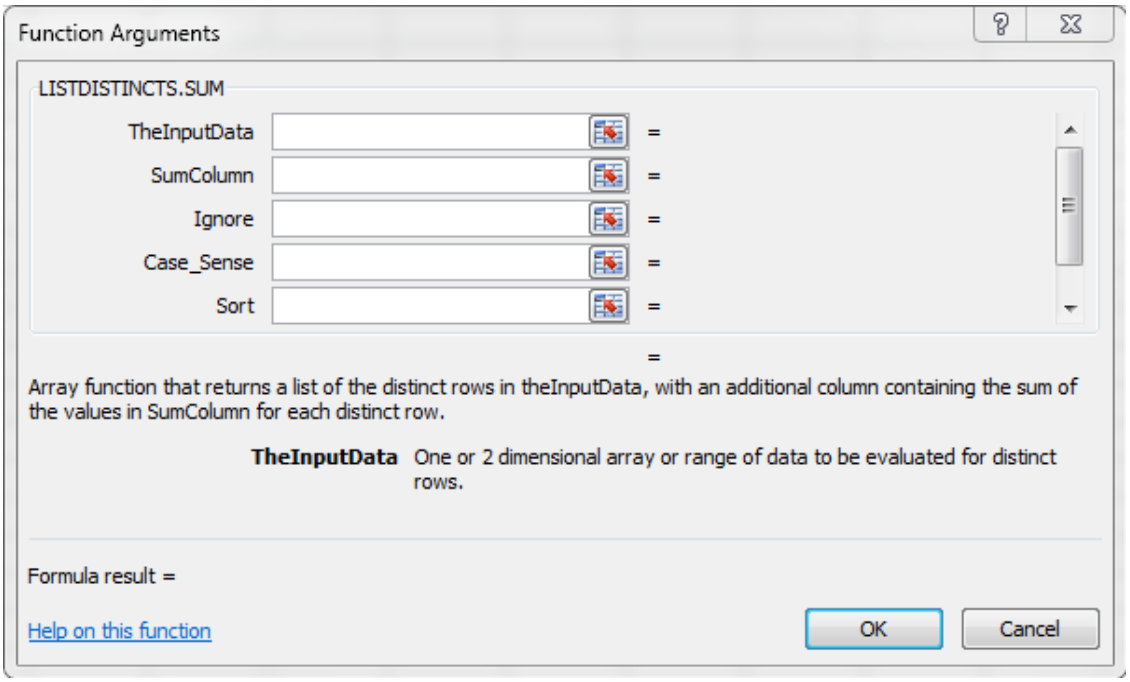
Nesting SpeedTools functions using the Excel Function Wizard

If you want to embed SpeedTools functions inside other functions you need to use the Excel Function Wizard instead of the SpeedTools Function Library Toolbar or Ribbon Group.

In the formula bar select the point in the existing formula where you want to embed the function and click the Excel Formula wizard button.

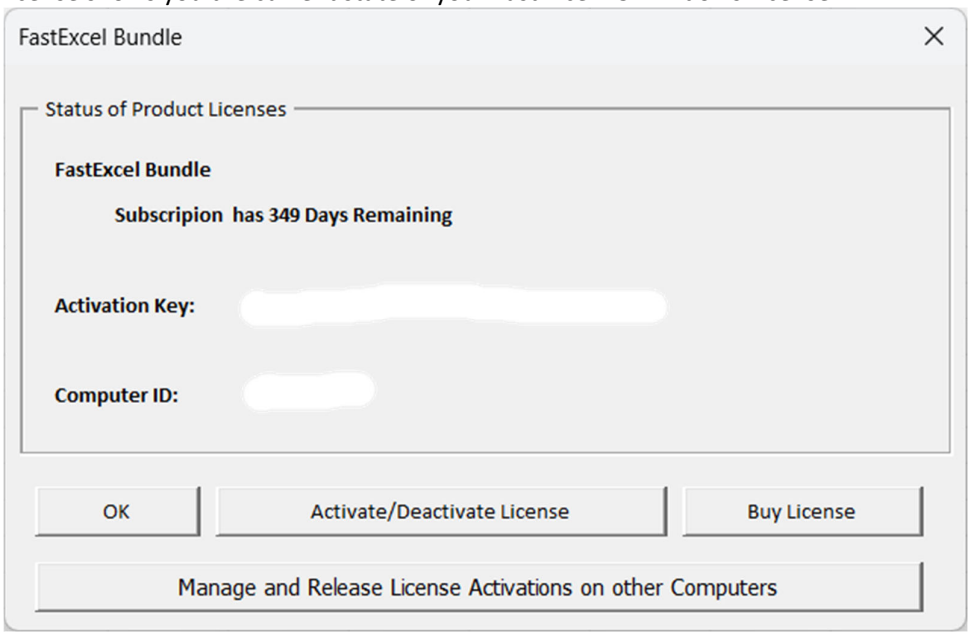
Excel Wizard Help

All the FastExcel SpeedTools functions are available from the Excel Function Wizard. This includes providing a short description of the function, its arguments and further Help through the Help button on the Function Wizard.



V5 License Management

License shows you the current state of your FastExcel V5 Windows License.



Buy a license

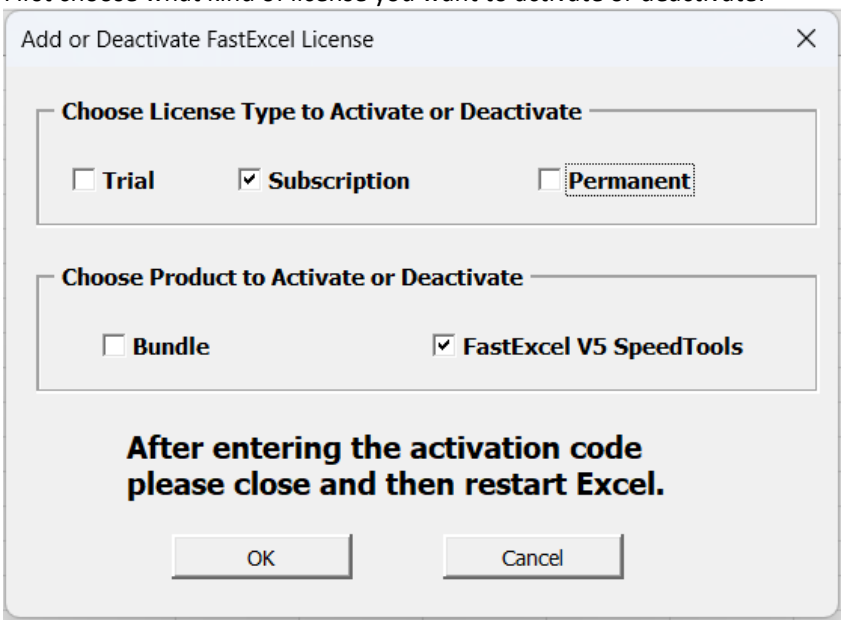
Links to the FastExcel Store.

Manage and Release License Activations on other computers

Links to the Decision Models QLM Customer Site where you can manage your FastExcel Licenses and deactivate licenses on other computers.

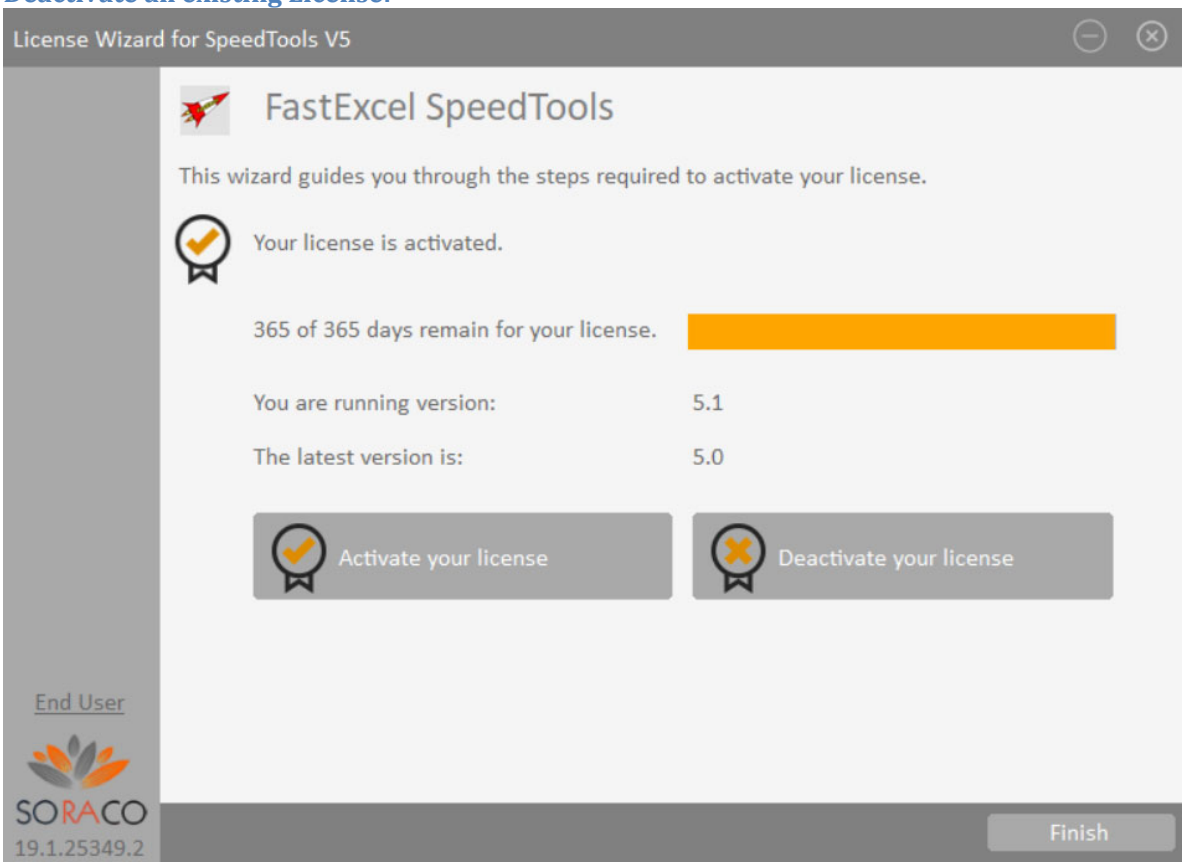
Activate/Deactivate License –

First choose what kind of license you want to activate or deactivate:

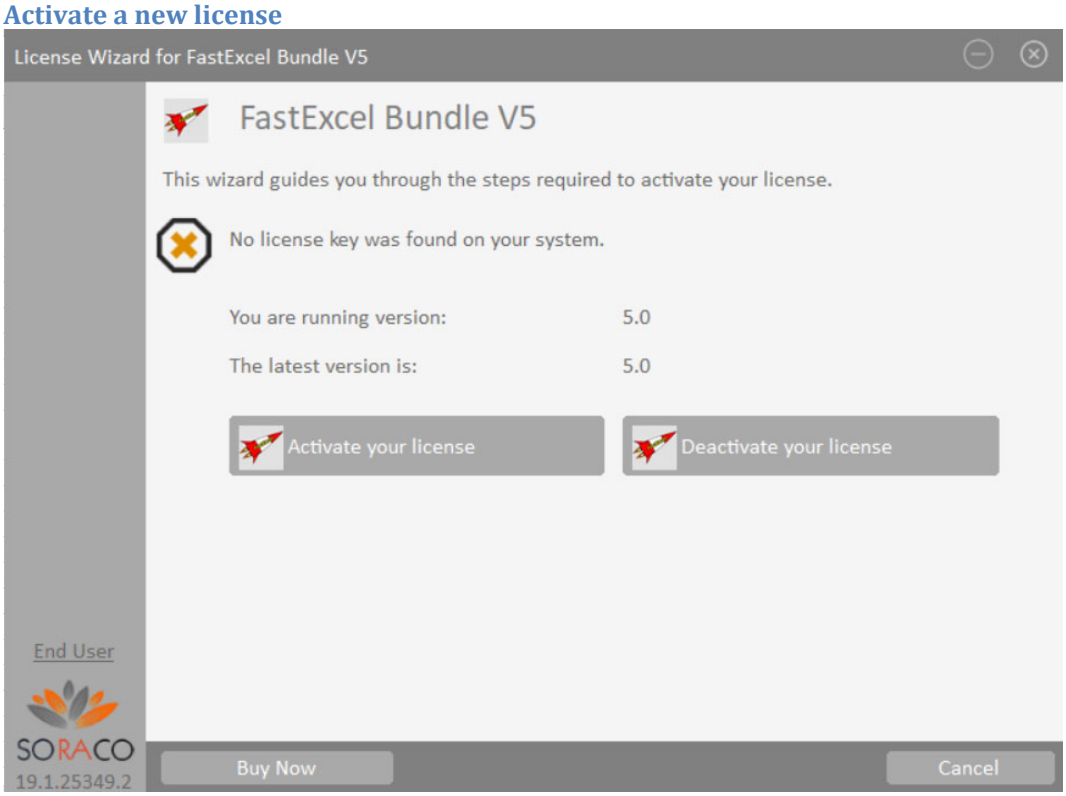


Clicking Ok will launch the QLM License Wizard:

Deactivate an existing License:



Activate a new license



SpeedTools Functions by Category

Name	SpeedTools Category	Excel Category
ACCUMULATE	Array	Math & Trig
ACOUNTIFS	Filters	Statistical
ALL	Logical	Logical
AMATCH2	Lookups	Lookup & Reference
AMATCHES2	Lookups	Lookup & Reference
AMATCHNTH	Lookups	Lookup & Reference
AND.CELLs	Logical	Logical
AND.COLS	Logical	Logical
AND.ROWS	Logical	Logical
ANTIJOIN	Lookups	Lookup & reference
ANY	Logical	Logical
ASUMIFS	Filters	Statistical
AVERAGECOLS	Array	Math & Trig
AVERAGEROWS	Array	Math & Trig
AVLOOKUP2	Lookups	Lookup & Reference
AVLOOKUPNTH	Lookups	Lookup & Reference
AVLOOKUPS2	Lookups	Lookup & Reference
CalcSeqCountRef	Information	Information
CalcSeqCountSet	Information	Information
CalcSeqCountVol	Information	Information
Case.AMATCH2	Lookups	Lookup & Reference
Case.AMATCHES2	Lookups	Lookup & Reference
Case.AMATCHNTH	Lookups	Lookup & Reference
Case.AVLOOKUP2	Lookups	Lookup & Reference
Case.AVLOOKUPNTH	Lookups	Lookup & Reference
Case.AVLOOKUPS2	Lookups	Lookup & Reference
Case.VSORTC	Sorting	Math & Trig
Case.VSORTC.INDEX	Sorting	Math & Trig
COL.ARRAY	Array	Lookup & Reference
COMPARE	Text	Text
COMPARE.LISTS	Lookups	Lookup & Reference
CONCAT.RANGE	Text	Text
CORKSCREW	Array	Math & Trig
COUNTCOLS	Array	Math & Trig
COUNTCOLS2	Information	Information
COUNTCONTIGCOLS2	Information	Information
COUNTCONTIGROWS2	Information	Information
COUNTDISTINCTS	Filters	Statistical
COUNTDUPES	Filters	Statistical
COUNTROWS	Array	Math & Trig
COUNTROWS2	Information	Information

COUNTUSEDCOLS2	Information	Information
COUNTUSEDROWS2	Information	Information
DIFF	Array	Math & Trig
EVAL2	Lookups	Lookup & Reference
FILTER.IFS	Filters	Lookup & Reference
FILTER.MATCH	Filters	Lookup & Reference
FILTER.SORTED	Filters	Lookup & Reference
FILTER.VISIBLE	Filters	Lookup & Reference
FILTERX	Filters	Lookup & Reference
GETMEM	Lookups	Lookup & Reference
GINICOEFF	Math	Statistical
GROUPS	Text	Text
HASFORMULA2	Information	Information
HDR.HSTACK	Array	Lookup & Reference
HDR.VSTACK	Array	Lookup & Reference
HSTACK & HSTACKF	Array	Lookup & Reference
HTOTALS	Array	Math & Trig
IFERRORX	Logical	Logical
ISLIKE2	Text	Text
INSPAN	Filters	Lookup & Reference
JOIN	Lookups	Lookup & Reference
LINTERP2D	Math	Math & Trig
LISTDISTINCTS	Filters	Statistical
LISTDISTINCTS.AVG	Filters	Statistical
LISTDISTINCTS.COUNT	Filters	Statistical
LISTDISTINCTS.SUM	Filters	Statistical
MATCH.SPAN	Filters	Lookup & Reference
MATCHES	Lookups	Lookup & Reference
MEMLOOKUP	Lookups	Lookup & Reference
MEMMATCH	Lookups	Lookup & Reference
MISSMATCHES	Lookups	Lookup & reference
MOVAVG	Array	Math & Trig
NONE	Logical	Logical
OR.CELLS	Logical	Logical
OR.COLS	Logical	Logical
OR.ROWS	Logical	Logical
PAD.ARRAY	Array	Lookup & Reference
PAD.TEXT	Text	Text
PREVIOUS	Lookups	Lookup & Reference
REPEAT	Array	Lookup & Reference
RESIZE	Array	Lookup & Reference
REVERSE.ARRAY	Array	Lookup & Reference
REVERSE.TEXT	Text	Text

Rgx.AMATCH2	Lookups	Lookup & Reference
Rgx.AMATCHES2	Lookups	Lookup & Reference
Rgx.AMATCHNTH	Lookups	Lookup & Reference
Rgx.AVLOOKUP2	Lookups	Lookup & Reference
Rgx.AVLOOKUPNTH	Lookups	Lookup & Reference
Rgx.AVLOOKUPS2	Lookups	Lookup & Reference
Rgx.Case.AMATCH2	Lookups	Lookup & Reference
Rgx.Case.AMATCHES2	Lookups	Lookup & Reference
Rgx.Case.AMATCHNTH	Lookups	Lookup & Reference
Rgx.Case.AVLOOKUP2	Lookups	Lookup & Reference
Rgx.Case.AVLOOKUPNTH	Lookups	Lookup & Reference
Rgx.Case.AVLOOKUPS2	Lookups	Lookup & Reference
Rgx.COUNTIF	Filters	Math & Trig
Rgx.FIND	Text	Text
Rgx.ISLIKE	Text	Text
Rgx.LEN	Text	Text
Rgx.MID	Text	Text
Rgx.SUBSTITUTE	Text	Text
Rgx.SUMIF	Filters	Math & Trig
ROW.ARRAY	Array	Lookup & Reference
SETMEM	Lookups	Lookup & Reference
SLICES	Array	Lookup & Reference
SPAN	Filters	Lookup & Reference
SPLICECELLS	Array	Lookup & Reference
SPLICECOLS	Array	Lookup & Reference
SPLICEROWS	Array	Lookup & Reference
SPLIT.TEXT	Text	Text
SUMCOLS	Array	Math & Trig
SUMROWS	Array	Math & Trig
TOTALS	Array	Math & Trig
UNPIVOT	Array	Lookup & Reference
VECTOR	Array	Lookup & Reference
VLINTERP2	Math	Math & Trig
VSORTB	Sorting	Math & Trig
VSORTB.INDEX	Sorting	Math & Trig
VSORTC	Sorting	Math & Trig
VSORTC.INDEX	Sorting	Math & Trig
VSTACK & VSTACKF	Array	Lookup & Reference

SpeedTools Functions Properties

Name	Multi-Thread	Volatile	Input Array	Input Range	Output
ACCUMULATE	Y	N	Y	Y	Y
ACOUNTIFS	Y	N	Y	Y	N
ALL	Y	N	Y	Y	N
AMATCH2	Y	N	Y	Y	Y
AMATCHES2	Y	N	Y	Y	Y
AMATCHNTH	Y	N	Y	Y	Y
AND.CELLs	Y	N	Y	Y	Y
AND.COLS	Y	N	Y	Y	Y
AND.ROWS	Y	N	Y	Y	Y
ANTIJOIN	Y	N	Y	Y	Y
ANY	Y	N	Y	Y	N
ASUMIFS	Y	N	Y	Y	N
AVERAGECOLS	Y	N	Y	Y	Y
AVERAGEROWS	Y	N	Y	Y	Y
AVLOOKUP2	Y	N	Y	Y	Y
AVLOOKUPNTH	Y	N	Y	Y	Y
AVLOOKUPS2	Y	N	Y	Y	Y
CalcSeqCountRef	N	N	N	Single Cell	N
CalcSeqCountSet	N	N	N	Single Cell	N
CalcSeqCountVol	N	Y	N	Single Cell	N
Case.AMATCH2	Y	N	Y	Y	Y
Case.AMATCHES2	Y	N	Y	Y	Y
Case.AMATCHNTH	Y	N	Y	Y	Y
Case.AVLOOKUP2	Y	N	Y	Y	Y
Case.AVLOOKUPNTH	Y	N	Y	Y	Y
Case.AVLOOKUPS2	Y	N	Y	Y	Y
Case.VSORTC	Y	N	Y	Y	Y
Case.VSORTC.INDEX	Y	N	Y	Y	Y
COL.ARRAY	Y	N	Y	Y	Y
COMPARE	Y	N	Scalar	Single Cell	N
COMPARE.LISTS	Y	N	Y	Y	Y
CONCAT.RANGE	Y	N	Y	Y	N
CORKSCREW	Y	N	Y	Y	Y
COUNTCOLS	Y	N	Y	Y	Y
COUNTCOLS2	N	N	N	Y	N
COUNTCONTIGCOLS2	N	N	N	Y	N
COUNTCONTIGROWS2	N	N	N	Y	N
COUNTDISTINCTS	Y	N	Y	Y	N
COUNTDUPES	Y	N	Y	Y	N
COUNTROWS	Y	N	Y	Y	Y
COUNTROWS2	N	N	N	Y	N

COUNTUSEDCOLS2	N	N	N	Y	N
COUNTUSEDROWS2	N	N	N	Y	N
DIFF	Y	N	Y	Y	Y
EVAL2	N	Y	Scalar	Single Cell	Y
FILTER.IFS	Y	N	Y	Y	Y
FILTER.MATCH	Y	N	Y	Y	Y
FILTER.SORTED	Y	N	Y	Y	Y
FILTER.VISIBLE	N	Y	N	Y	Y
FILTERX	Y	N	Y	Y	Y
GETMEM	Y	N	Y	Y	Y
GINICOEFF	Y	N	Y	Y	Y
GROUPS	Y	N	Y	Y	Y
HASFORMULA2	N	N	N	Y	N
HDR.HSTACK	Y	N	Y	Y	Y
HDR.VSTACK	Y	N	Y	Y	Y
HSTACK & HSTACKF	Y	N	Y	Y	Y
HTOTALS	Y	N	Y	Y	Y
IFERRORX	Y	N	Y	Y	Y
INSPAN	Y	N	Y	Y	Y
ISLIKE2	Y	N	Y	Y	Y
JOIN	Y	N	Y	Y	Y
LINTERP2D	Y	N	Y	Y	Y
LISTDISTINCTS	Y	N	Y	Y	Y
LISTDISTINCTS.AVG	Y	N	Y	Y	Y
LISTDISTINCTS.COUNT	Y	N	Y	Y	Y
LISTDISTINCTS.SUM	Y	N	Y	Y	Y
MATCHES	Y	N	Y	Y	Y
MATCH.SPAN	Y	N	Y	Y	Pair
MEMLOOKUP	Y	N	Y	Y	N
MEMMATCH	Y	N	Y	Y	N
MISSMATCHES	Y	N	Y	Y	Y
MOVAVG	Y	N	Y	Y	Y
NONE	Y	N	Y	Y	N
OR.CELLS	Y	N	Y	Y	Y
OR.COLS	Y	N	Y	Y	Y
OR.ROWS	Y	N	Y	Y	Y
PAD.ARRAY	Y	N	Y	Y	Y
PAD.TEXT	Y	N	Y	Y	Y
PREVIOUS	N	Optional	N	Y	Y
REPEAT	Y	N	Y	Y	Y
RESIZE	Y	N	Y	Y	Y
REVERSE.ARRAY	Y	N	Y	Y	Y
REVERSE.TEXT	Y	N	Y	Y	Y

Rgx.AMATCH2	Y	N	Y	Y	Y
Rgx.AMATCHES2	Y	N	Y	Y	Y
Rgx.AMATCHNTH	Y	N	Y	Y	Y
Rgx.AVLOOKUP2	Y	N	Y	Y	Y
Rgx.AVLOOKUPNTH	Y	N	Y	Y	Y
Rgx.AVLOOKUPS2	Y	N	Y	Y	Y
Rgx.Case.AMATCH2	Y	N	Y	Y	Y
Rgx.Case.AMATCHES2	Y	N	Y	Y	Y
Rgx.Case.AMATCHNTH	Y	N	Y	Y	Y
Rgx.Case.AVLOOKUP2	Y	N	Y	Y	Y
Rgx.Case.AVLOOKUPNTH	Y	N	Y	Y	Y
Rgx.Case.AVLOOKUPS2	Y	N	Y	Y	Y
Rgx.COUNTIF	Y	N	Y	Y	N
Rgx.FIND	Y	N	Y	Y	Y
Rgx.ISLIKE	Y	N	Y	Y	Y
Rgx.LEN	Y	N	Y	Y	Y
Rgx.MID	Y	N	Y	Y	Y
Rgx.SUBSTITUTE	Y	N	Y	Y	Y
Rgx.SUMIF	Y	N	Y	Y	N
ROW.ARRAY	Y	N	Y	Y	Y
SETMEM	Y	N	Y	Y	Y
SLICES	Y	N	Y	Y	Y
SPAN	Y	N	Y	Y	Y
SPLICECELLS	Y	N	Y	Y	Y
SPLICECOLS	Y	N	Y	Y	Y
SPLICEROWS	Y	N	Y	Y	Y
SPLIT.TEXT	Y	N	Y	Y	Y
SUMCOLS	Y	N	Y	Y	Y
SUMROWS	Y	N	Y	Y	Y
TOTALS	Y	N	Y	Y	Y
UNPIVOT	Y	N	Y	Y	Y
VECTOR	Y	N	Y	Y	Y
VLINTERP2	Y	N	Y	Y	Y
VSORTB	Y	N	Y	Y	Y
VSORTB.INDEX	Y	N	Y	Y	Y
VSORTC	Y	N	Y	Y	Y
VSORTC.INDEX	Y	N	Y	Y	Y
VSTACK & VSTACKF	Y	N	Y	Y	Y

Array totalling and arithmetic functions

ACCUMULATE Function

Accumulates carry-forward values column-by-column or row-by-row, optionally adding a set of constants and factors times the brought-forward value to the brought-forward value to produce the next carry-forward value.
The function is designed to allow this type of calculation with dynamic arrays without the use of circular references/iterative calculation.

This function is based on a suggestion made by Peter Bartholomew.

The ACCUMULATE function calculates recurrence relationships of the form:

$$U_{n+1} = a_n + (1 + b_n)U_n$$

Where **a** is the AddValues parameter and **b** is the Factors parameter

ACCUMULATE is a non-volatile multi-threaded array function

ACCUMULATE Syntax

ACCUMULATE([Range_Array] [,StartValue] [,Factors] [,Closing])

Range_Array (Optional. Default zero)

A range or array of values to be added to the closing balances.

StartValue (Optional. Default zero)

The initial opening value.

Factors (Optional. Default zero)

A range or array giving the multiplying factors to be applied to the opening values and the result added to the closing value.
If only a single factor is given it will be propagated across or down.
The results will be added to the closing balances.

Closing (Optional. Default TRUE)

True returns the closing values of the accumulation, False returns the opening values

ACCUMULATE examples

Cumulative Sum

	A	B	C	D	E
1	74	96	69	4	65
2	=ACCUMULATE(A1:E1)				
3	74	170	239	243	308

Cash Balance with Interest

Starting with a value of 100, an interest rate of 5% each period and a year by year net cashflow, this example uses two ACCUMULATE functions to separately calculate the opening cash balance and the closing cash balance.

G	H	I	J	K	L
Interest Rate	5%	5%	5%	5%	5%
Cash Flow	15.40	10.75	14.14	15.20	12.25
	2019	**2020**	**2021**	**2022**	**2023**
=ACCUMULATE($H$2:$L$2,100,$H$1:$L$1,FALSE)	100.00	120.40	137.17	158.17	181.28
=ACCUMULATE($H$2:$L$2,100,$H$1:$L$1)	120.40	137.17	158.17	181.28	202.59

DIFF Function

DIFF calculates the difference between succeeding numbers in a range or array.
The calculation can be done recursively as many times as desired (Order).

DIFF returns an array of differences which may be shorter than the set of values from the input range or array. The number of terms returned is dependent on the chosen values of the DIFF parameters: for example, with a lag of 1 and an order of 1 and a total of 10 numbers 9 differences will be returned.

Options are provided to control the interval between the items to be differenced (Lag) and the number of times the difference calculation will be done on the input data (Order).
The initial value for DIFF can either be the start value in the range or array (or the last value with negative Lag), or a separately supplied Start Value.

The differences returned may optionally be aligned with the corresponding positive terms in the difference. In this case the values returned will be padded with using the value of the Align parameter to match the number of values in the input Range_Array.

DIFF is a non-volatile multi-threaded array function.

DIFF Syntax

DIFF (Range_Array [,StartValue] [,Lag] [,Order] [,Align] [,Down])

Range_Array

The Values to be DIFFed. Can be a Range or an Array, a Column or a Row or a range of rows and columns. If any non-numeric or empty values are found in Range-Array DIFF will return #Value.

StartValue (Optional.)

If the StartValue parameter is supplied then it's value will be used as the first value of the series of values for positive Lag or as the last value in the series for negative Lag.

If StartValue is not supplied the first or last value (depending on the sign of Lag) in Range_Array will be used as the start value.

If Range_Array is a 2-dimensional set of values then StartValue must be a corresponding single row or column of start values.

Lag (Optional. Default 1)

A positive or negative integer giving the interval between the values to be DIFFed.

Positive values for Lag give differences going forward.
A Value of 1 returns the second value minus the first, the third value minus the second etc. A Value of 2 returns the third value minus the first, the fourth value minus the second etc.

Negative values for Lag give differences going backwards.
A Value of -1 returns the first value minus the second, the second value minus the third etc. A value of -2 returns the first value minus the third, the second value minus the fourth etc.

Order (Optional. Default 1)

A positive integer giving the number of times the difference operation will be done.

A value of 2 is equivalent to DIFF(DIFF(..)

Align (Optional. Default 0.0)

When Align is used DIFF will line up the difference results with the corresponding terms being differenced from in the input values.

With a Lag of +1 the result of the second value minus the first value will be in the second position.
With a Lag of -1 the result of the first value minus the second value will be in the first position.

When Align is set to FALSE the first calculated difference is always in the first position regardless of Lag, and the number of values in the output will be less than the count of values in Range_Array and StartValue.

The output will be padded as required at the start or end using whatever value is given for Align to ensure that the number of values in the output is the same as the number of values in the input.

Typical values for Align are 0.0 or “”.

Down (Optional.)

When Range_Array contains multiple rows and columns Down determines the direction of the difference calculation.

By default, if there are more rows than columns in Range_Array Down will be True, else False.

When specified Down overrides the default behaviour.

If Down is True then a separate set of differences will be calculated **for each column** in Range_Array.
If Down is False then a separate set of differences will be calculated **for each row** in Range_Array.

DIFF Examples

Difference of the Closing Balance with a Start Value of 100

	A	B	C	D	E	F	G
1							
2			**2019**	**2020**	**2021**	**2022**	**2023**
3	**Closing Balance**		120.40	137.17	158.17	181.28	202.59
4	**CashFlow**	**=DIFF($C$3:$G$3,100)**	20.40	16.77	21.00	23.11	21.31

In the image below:
Column J shows a DIFF of the data using all the defaults. The results appear starting in the second row. The default is Aligned =0.0 so the first result is 0.0 and the result of 9.0-1.0 = 8.0 is positioned in the same row as the 9.0

Column K shows a second-order difference of the data with Align=False. This is equivalent to DIFF(DIFF …))

Column M shows an Aligned Diff padded out with “” – the differences are in the same row as the first term in the difference.

Column O shows an Aligned Diff with a Lag of 2. The first result is the third value minus the first value. NA is used as the Align value to make it easier to see where the unavailable results are.

Column Q shows a Diff with a Lag of minus 2 and Align=False. The first result is the first value minus the third value.

I	J	K	L	M	N	O	P	Q
	Defaults	**Order=2, Align=FALSE**		**Align=""**		**Align="NA" Lag= plus 2**		**Align False, Lag= minus 2**
Data	**DIFF($I$3:$I$12)**	**DIFF($I$3:$I$12,,,2,FALSE)**		**DIFF($I$3:$I$12,,,,"")**		**DIFF($I$3:$I$12,,2,,"NA")**		**DIFF($I$3:$I$12,,-2,,FALSE)**
1.0	0.0	9.0				NA		-25.0
9.0	8.0	6.0		8.0		NA		-40.0
26.0	17.0	10.0		17.0		25.0		-56.0
49.0	23.0	6.0		23.0		40.0		-72.0
82.0	33.0	10.0		33.0		56.0		-88.0
121.0	39.0	6.0		39.0		72.0		-104.0
170.0	49.0	10.0		49.0		88.0		-120.0
225.0	55.0	6.0		55.0		104.0		-136.0
290.0	65.0			65.0		120.0		
361.0	71.0			71.0		136.0		

Differences, then differences of differences, then differences of differences of differences

Values	**DIFF Values with Order 1, 2 and 3**		
1	8	8	0
9	16	8	0
25	24	8	0
49	32	8	0
81	40	8	0
121	48	8	0
169	56	8	0
225	64	8	
289	72		
361			

MOVAVG Function

MOVAVG can calculate 6 different types of moving averages for a series:

- Simple moving averages (SMA)
- Central simple moving averages (CSMA)
- Cumulative moving averages (CUSMA)
- Weighted moving averages (WMA)
- Exponential moving averages (EMA)
- Double exponential moving averages (Brown's method) (DEMA)

A set of moving averages is calculated from the input data and returned as an array.
If the input data has more than one row and more than one column (multiple series of data), a set of moving averages is calculated for each series.
Options are provided to control the number of terms in the moving averages and the starting method for the exponential moving averages.

MOVAVG is a non-volatile multi-threaded array function.

MOVAVG Syntax

MOVAVG (Range_Array, N, MA_Method [, Down] [,StartExp])

Range_Array

The values to be used for the moving average calculations. Can be a Range or an Array, a Column or a Row or a range of rows and columns.
If any non-numeric values are found in Range-Array MOVAVG will return #Value.

N

For Simple, Central simple and Weighted moving averages defines the number of items to be used in the moving average.

For Cumulative moving averages N is ignored.

For Exponential moving averages N, controls the smoothing constant (Alpha) and hence the amount of smoothing used in the calculation.
Alpha is calculated as 2/(N+1) so if N=9 then Alpha =0.2
To calculate N for a given Alpha use N=2/Alpha-1 so for Alpha=0.1, N= 19

MA_Method

Controls the calculation method used for the moving averages.

1 = Simple Moving Average.
Returns the average of each set of N consecutive values. The value is returned at the position of the last value in each set: so if N=4 zero is returned in position 1,2 and 3 and the first moving average is in position 4 followed by the 4-term moving average for each successive set.

The 4-term moving average of 1,2,3,4 is (1+2+3+4)/4 = 2.5 in position 4.

2 = Central Simple Moving Average.
For some types data it is more appropriate to return the moving average in the central position of the set. For example, if N=5 the first moving average will be returned in position 3.
For even values of N the moving average is calculated as the average of the 2 central terms and is returned at the position of the first central term. For example, if N=4 then two 4-term moving averages will be calculated centered on both position 2 and 3, and the average of these two movong averages will be placed at position 2.

3 = Cumulative Moving Average.
This is calculated as the cumulative sum of the values so far divided by the number of values so far. The value of N is meaningless for a CUSMA and is ignored.

4 = **Weighted Moving Average**.
An ascending sequence of weights is applied to the values in each set of N values to give more emphasis to the more recent values. The weights used are an ascending series of integers, divided by the sum of the integers.

For an N of 4 the weights are 1,2,3,4 with each weight divided by SUM(1,2,3,4)=10
For values 5,6,7,8 the WMA of this 4-term group is calculated as
(5*1+6*2+7*3+8*4)/ (1+2+3+4) =7

5 = Exponential Moving Average.

Each term in an EMA is calculated as
EMA(t)= Alpha x(t) + (1-Alpha)* EMA(t-1)*
Where:

- t denotes the position in the data series
- x (t) is the value in the data series at that position.
- EMA(t) is the calculated exponentially smoothed value at position t
- Alpha is the smoothing constant determined from N. It must be between 0 and 1.

The higher the value of Alpha the more weight is given to recent values (and the lower the smoothing).

MOVAVG calculates Alpha from N using Alpha=2/(N+1),
so a higher value of N gives a greater degree of smoothing.

There are 2 empirical factors in the EMA calculation that need to be chosen by the user.

- What to use as the starting value. This is controlled by the value of StartExp.
- What value to use for Alpha. This is controlled by the value of N.

Note: When using EMA the value of N does not determine the number of terms to average, it only determines the vale of Alpha. EMA returns a calculated EMA value for every term in the data.

6 = Double Exponential Moving Average (Brown's method).

EMA tends to lag behind changes in the data and is therefore best suited to stationary data series (data has no overall trend).

DEMA adds a trend term by calculating both an EMA and an EMA of the EMA and then combining the two.

*S1(t) = Alpha * x(t) + (1-Alpha) * S1(t-1)*

*S2(t) = Alpha * S1(t) + (1-Alpha) * S2(t-1)*

DEMA(t) = 2 S1(t) – S2(t)*

*Trend(t) = (Alpha / (1-Alpha)) * (S1(t)-S2(t))*

Where:

- t denotes the position in the data series
- x (t) is the value in the data series at that position.
- S1 is the EMA of the data series x
- S2 is the EMA of the S1 EMA
- DEMA(t) is the calculated doubly exponentially smoothed value at position t

- Alpha is the smoothing constant determined from N. It must be between 0 and 1.
- Trend(t) is the trend term at position t

There are 2 empirical factors in the DEMA calculation that need to be chosen by the user.

- What to use as the starting values for S1(0) and S2(0). This is controlled by the value of StartExp.
- What value to use for Alpha. This is controlled by the value of N.

Note: When using DEMA the value of N does not determine the number of terms to average, it only determines the vale of Alpha. DEMA returns a calculated DEMA value for every term in the data.

Down (Optional. Default True)

This parameter is ignored when Range_Array is a single row or column.

When Range_Array contains multiple rows and columns Down determines the direction of the moving average calculation.
If Down is True then a separate series of moving averages will be calculated for each column in Range_Array.
If Down is False then a separate separate series of moving averages will be calculated for each row in Range_Array.

StartExp (Optional. Default 1)

This parameter is only used for EMA and DEMA calculations. It controls the way the starting values at position zero are calculated.

For large values of N and low values of the smoothing constant Alpha the initial values used for the exponential smoothing calculation can have a significant effect on the smoothed result.

StartExp=1

The first value in the data is used as the starting value:

EMA(1)=x(1)

S1(1)=x(1)

S2(1)=x(1)

StartExp=0

The first N values in the data are used to calculate the starting values.

StartExp = +ve

Where +ve is a positive integer. The first +ve values in the data are used to calculate the starting values.

The calculation when StartExp>=0 is:

EMA*(1) = Average of the first N or StartExp values in the data*

DEMA:

S1(1) = Average of the first N or StartExp values in the data

S2(1) = Average of the first N or StartExp S1 values

*DEMA(1) = 2 * S1(1) – S2(1)*

StartExp = -ve

When StartExp has a negative value the starting values at position zero for both DEMA and EMA are calculated by back-casting from the last values in the data, using the standard EMA and DEMA calculation methods.

MOVAVG Examples

These examples show the output and calculation methods for the 6 MOVAVG calculation methods.

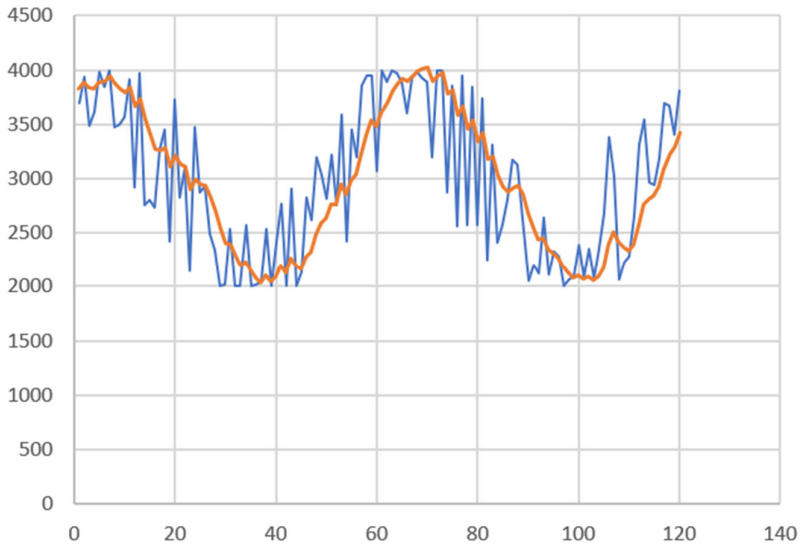
- Data is shown in grey.
- Result of the moving average function is shown in pale green.
- The calculation steps are shown in pale blue.

B	C	D	E	F	G	H
Data	1	2	3	4	5	6
SMA N=3	0	0	2	3	4	5
MOVAVG($C$2:$H$2,3,1)		(1+2+3)/3	(2+3+4)/3	(3+4+5)/3	(4+5+6)/3	
CMA N=4	0	3	4	0	0	0
MOVAVG($C$2:$H$2,4,2)		((1+2+3+4)/4+(2+3+4+5)/4)/2				
		((2+3+4+5)/4+(3+4+5+6)/4)/2				
CUSMA N=?	1	1.5	2	2.5	3	3.5
MOVAVG($C$2:$H$2,2,3)	1 (1+2)/2	(1+2+3)/3	(1+2+3+4)/4 ...			
WMA N=4	0	0	0	3	4	5
MOVAVG($C$2:$H$2,4,4)			(1*1+2*2+3*3+4*4)/10			
Weights are 1,2,3, ... N				(2*1+3*2+4*3+5*4)/10		
					(3*1+4*2+5*3+6*4)/10	
EMA N=4, StartExp=1	1	1.4	2.04	2.824	3.6944	4.61664
Alpha =2/(N+1)=0.4	1 (1-0.4)*1+0.4*2	(1-0.4)*1.4+0.4*3	(1-0.4)*2.04+0.4*4	(1-0.4)*2.824+0.4*5	(1-0.4)*3.6944+0.4*6	
DEMA N=4, StartExp=1	1	1.64	2.568	3.6112	4.68896	5.76672
S1=EMA(Data)	1	1.4	2.04	2.824	3.6944	4.61664
S2=EMA(S1)	1	1.16	1.512	2.0368	2.69984	3.46656
DEMA=2*S1-S2	1	1.64	2.568	3.6112	4.68896	5.76672
DEMA N=4, StartExp= -1	1.23328	1.4479616	2.25357312	3.303021568	4.432339558	5.569719706
Backcast start values						

This example uses DEMA with N=18 and back-casting for the starting values:

=MOVAVG(C6#,18,6,,-1)

Data is in blue and the output from MOVAVG is in orange.



Higher values of N would result in a smoother curve but more lag.

TOTALS & AGGREGATES Functions

Floating or fixed column totals: Totals each column of the Table_Range/Expression.
Floating produces a copy of the Table_Range/Expression with a horizontal array of the totals of the columns appended at the bottom.
Fixed produces a fixed position horizontal array of the totals of the columns without a copy of the table/range.
Dynamically adjusts to the number of rows and columns in Table_Range
TOTALS and AGGREGATES are multi-threaded, non-volatile array functions.

AGGREGATES is an alias of TOTALS

TOTALS/AGGREGATES Syntax

TOTALS (Table_Range [, Exclude_Cols] [,TotalTypes], [Fixed])

AGGREGATES (Table_Range [, Exclude_Cols] [,TotalTypes], [Fixed])

Table_Range

Array, Table, Range or Spill reference whose columns are to be totalled.

Exclude_Cols (Optional. Default no columns will be excluded)

Optional. If omitted all columns will be totalled.
A list (array or range) identifying the columns that are **not** to be totalled, given as one of:

- If Table_Range is not an array: One or more ranges intersecting columns inTable_Range. Multiple disjoint ranges must be enclosed in ( ) – for example (A1:B1,D1) results in columns 1,2 and 4.
- Alphabetic labels are used as lookup values in the first row of Table_Range to find the excluded columns. These labels can be supplied in an array or a range.
- Relative column numbers are used directly.
  These column numbers can be supplied in an array or range.

  For example {2,4} excludes the second and fourth column of the Table_Range.
  Negative numbers work from right to left so {-1, -2) excludes the last two columns of the Table_Range.

  ***Zero acts as an include operator*.**
  {1,0,5} results in Columns 1 through 5 inclusive
  {2,0,-2} results in the second column and all columns up and including the second-to-last column
  {1,5} results in Column 1 and Column 5 only
  {5,0} results in Column 5 and all Columns after Column 5
  {0,5} results in the first 5 Columns

TotalTypes (Optional, Default 9 =SUM)

An array/range of types to control the type of totalling to be done. Allows for different total types for different columns.

- 1 = Average
- 2 = Count
- 3 = CountA
- 4 = Max
- 5 = Min
- 6 = Product
- 7 = Stdev
- 8 = stdevp

- 9 = Sum
- 10 = Var
- 11 = Varp
- 12 = Median
- 13 = Mode.Sngl

Empty, Zero, Missing or non-numeric values within the array/range will be treated as 9=SUM.
If only a single TotalType is given it will be used for all columns.

Fixed (Optional. Default False)

If TRUE the Totals will be in a fixed but spilling across location starting from the cell containing the function, if FALSE the Totals will be floating as the last row of whatever size the dynamic array is.

TOTALS Example:

=TOTALS(SORT(FILTER(Sales,Sales[Region]=M6),6,-1),9,{1,2,3,5})

The TOTALS function is wrapped around a dynamic array expression, so that as the number of rows spilled by the dynamic array expression changes the totals are always the last row.

The TotalType is 9 which means SUM.

Columns 1,2,3 and 5 are excluded from the totals.

Invoice	Product	Region	Units	Price	Revenue
175	Charging Station	Kansas	4	53.67	214.68
122	Charging Station	Kansas	4	52.88	211.52
19	Charging Station	Kansas	4	51.91	207.64
81	Charging Station	Kansas	2	51.87	103.74
18	Digital Projector	Kansas	1	87.14	87.14
277	Digital Projector	Kansas	1	85.9	85.90
298	Gaming Console	Kansas	5	15.56	77.80
182	Home Automation Hub	Kansas	2	24.78	49.56
103	Gaming Console	Kansas	1	13.9	13.90
			30		1,560.02

Accessing the floating Totals from a formula.

You can use the SLICES function to access the floating Totals Row. If the dynamic array formula is in K9 then

=SLICES(K9#,-1) gives the complete total row (-1 means the first row at the bottom),

and =SLICES(K9#,-1,-1) gives 1560.02, the last column in the last row.

To calculate **Yield = Total Revenue/Total Units** you could use the formula

= SLICES(K9#,-1,-1)/SLICES(K9#,-1,-3)

Adding a Total heading to the Totals row

You can use the SPLICECELLS function to replace cells in a dynamic array.

So to add the word “Totals” into the floating totals row wrap the TOTALS formula in

SPLICECELLS(total formula,-1,1,”Totals”)

The -1 specifies the last row of the dynamic array (which is the totals row) and the 1 specifies the first column.

245	Gaming Console	Maine	4	13.13	52.52
Totals			26		862.48

HTOTALS Function

Floating or fixed row totals: Totals each row of the Table_Range/Expression.
Floating produces a copy of the Table_Range/Expression with a vertical array of the totals of the rows appended at the right.
Fixed produces a fixed position vertical array of the totals of the rows without a copy of the table/range.
Dynamically adjusts to the number of rows and columns in Table_Range
HTOTALS is a multi-threaded, non-volatile array function.

HTOTALS Syntax

HTOTALS (Table_Range [, Exclude_Rows] [,TotalTypes] [,Fixed])

Table_Range

Array, Table, Range or Spill reference whose columns are to be totalled.

Exclude_Rows (Optional. Default no rows will be excluded)

Optional. If omitted all rows will be totalled.
A list (array or range) identifying the rows that are **not** to be totalled, given as one of:

- If Table_Range is not an array: One or more ranges intersecting rows in Table_Range. Multiple disjoint ranges must be enclosed in ( ) – for example (A1:B2,A4) results in rows 1,2 and 4.
- Alphabetic labels are used as lookup values in the first column of Table_Range to find the rows to exclude. These labels can be supplied in an array or a range.
- Relative row numbers are used directly.
  These row numbers can be supplied in an array or range.

  For example {2,4} excludes the second and fourth row of the Table_Range.
  Negative numbers work from bottom to top so {-1, -2) excludes the last two rows of the Table_Range.

  Zero acts as an include operator.
  {1,0,5} results in rows 1 through 5 inclusive
  {1,5} results in row 1 and row 5 only
  {5,0} results in row 5 and all rows after row 5
  {0,5} results in the first 5 rows

TotalTypes (Optional, Default 9 =SUM)

An array/range of types to control the type of totalling to be done. Allows for different total types for different columns.

- 1 = Average
- 2 = Count
- 3 = CountA
- 4 = Max
- 5 = Min
- 6 = Product
- 7 = Stdev
- 8 = stdevp
- 9 = Sum
- 10 = Var
- 11 = Varp
- 12 = Median
- 13 = Mode.Sngl

Empty, Zero, Missing or non-numeric values within the array/range will be treated as 9=SUM.
If only a single TotalType is given it will be used for all rows.

HTOTALS and TOTALS can be nested: =HTOTALS(TOTALS(MyData)) gives a row of totals for myData at the bottom, a column of totals for MyData at the right, and a grand total in the bottom right corner.

Fixed (Optional. Default False)

Controls whether the Totals will be in a fixed but spilling across location starting from the cell containing the function, or will be floating as the last row of whatever size the dynamic array is.

SUMCOLS Function

Sums each column of the Table_Range to produce a horizontal array of the sums of the columns.
Dynamically adjusts to the number of rows and columns in Table_Range:
use SUMCOLS when you want non-floating totals for dynamic arrays.
SUMCOLS is a multi-threaded, non-volatile array function.

SUMCOLS Syntax

SUMCOLS (Table_Range [, Exclude_Cols])

Table_Range

Array, Table, Range or Spill reference whose columns are to be summed.

Exclude_Cols (Optional. Default no columns will be excluded)

Optional. If omitted all columns will be summed.
A list (array or range) identifying the columns **that are not to be summed**, given as one of:

- If Table_Range is not an array: One or more ranges intersecting columns in Table_Range. Multiple disjoint ranges must be enclosed in ( ) – for example (A1:B1,D1) results in columns 1,2 and 4.
- Alphabetic labels are used as lookup values in the first row of Table_Range to find the columns. These labels can be supplied in an array or a range.
- Relative column numbers are used directly.
  These column numbers can be supplied in an array or range.

  For example {2,4} gets the second and fourth column of the Table_Range.
  Negative numbers work from right to left so {-1, -2) gets the last two columns of the Table_Range.

  ***Zero acts as an include operator*.**
  {1,0,5} results in Columns 1 through 5 inclusive
  {1,5} results in Column 1 and Column 5 only
  {5,0} results in Column 5 and all Columns after Column 5
  {0,5} results in the first 5 Columns

SUMROWS Function

Sums each row of the Table_Range to produce a vertical array of the sums of the rows.
Dynamically adjusts to the number of rows and columns in Table_Range:
use SUMROWS when you want non-floating totals for dynamic arrays.
SUMROWS is a multi-threaded, non-volatile array function.

SUMROWS Syntax

SUMROWS (Table_Range [, Exclude_Rows])

Table_Range

Array, Table, Range or Spill reference whose rows are to be summed.

Exclude_Rows (Optional. Default no rows will be excluded)

Optional. If omitted all rows will be summed.
A list (array or range) identifying the rows that are **not** to be summed, given as one of:

- If Table-Range is not an array: One or more ranges intersecting rows in Table_Range. Multiple disjoint ranges must be enclosed in ( ) – for example (A1:A2,A4) results in rows 1,2 and 4.
- Alphabetic labels are used as lookup values in the first column of Table_Range to find the rows. These labels can be supplied in an array or a range.
- Relative row numbers are used directly.
  These row numbers can be supplied in an array or range.

  For example {2,4} gets the second and fourth row of the Table_Range.
  Negative numbers work from the bottom upwards so {-1, -2) gets the last two rows of the Range_Array.

  Zero acts as an include operator.
  {1,0,5} results in rows 1 through 5 inclusive
  {1,5} results in row 1 and row 5 only
  {5,0} results in row 5 and all rows after row 5
  {0,5} results in the first 5 rows

SUMCOLS and SUMROWS example

In this example the totals for the dynamic array are fixed in position above and to the left of the dynamic array area, which starts in cell **L9**. This makes it easy to create % totals by referencing the dynamic array totals created by SUMCOLS and SUMROWS.
The column of totals is created by the formula **=SUMROWS(L9#)** in cell **J9**
The row of totals is created by the formula **=SUMCOLS(L9#)** in cell **L6**

	Total Value					
=SUM(J9#)	**7685002**	=SUMCOLS(L9#)	**2271436**	**18552**	**3348717**	**2046297**
		=L6#/$J$6	**29.6%**	**0.2%**	**43.6%**	**26.6%**
	=SUMROWS(L9#)	=J9#/$J$6	**ABC**	**CRW**	**DEF**	**XYZ**
Central	**2270394**	**29.5%**	766448	0	773927	730019
East	**3774400**	**49.1%**	860769	18552	2055308	839771
North	**21438**	**0.3%**	0	0	0	21438
West	1618770	**21.1%**	644219	0	519482	455069

AVERAGECOLS Function

Averages each column of the Table_Range to produce a horizontal array of the averages of the columns.
Only numbers are averaged: logical, text and empty cells are excluded
Dynamically adjusts to the number of rows and columns in Table_Range:
use AVERAGECOLS when you want non-floating totals for dynamic arrays.
AVERAGECOLS is a multi-threaded, non-volatile array function.

AVERAGECOLS Syntax

AVERAGECOLS (Table_Range [, Exclude_Cols])

Table_Range

Array, Table, Range or Spill reference whose columns are to be averaged.

Exclude_Cols (Optional. Default no columns will be excluded)

Optional. If omitted all columns will be averaged.
A list (array or range) identifying the columns that are **not** to be averaged, given as one of:

- If Table_Range is not an array: One or more ranges intersecting columns in Table_Range. Multiple disjoint ranges must be enclosed in ( ) – for example (A1:B1,D1) results in columns 1,2 and 4.
- Alphabetic labels are used as lookup values in the first row of Table_Range to find the columns. These labels can be supplied in an array or a range.
- Relative column numbers are used directly.
  These column numbers can be supplied in an array or range.

  For example {2,4} gets the second and fourth column of the Table_Range.
  Negative numbers work from right to left so {-1, -2) gets the last two columns of the Table_Range.

  Zero acts as an include operator.
  {1,0,5} results in Columns 1 through 5 inclusive
  {1,5} results in Column 1 and Column 5 only
  {5,0} results in Column 5 and all Columns after Column 5
  {0,5} results in the first 5 Columns

AVERAGEROWS Function

Averages each row of the Table_Range to produce a vertical array of the averages of the rows. Only numbers are averaged: logical, text and empty cells are excluded
Dynamically adjusts to the number of rows and columns in Table_Range:
use AVERAGEROWS when you want non-floating totals for dynamic arrays.
AVERAGEROWS is a multi-threaded, non-volatile array function.

AVERAGEROWS Syntax

AVERAGEROWS (Table_Range [, Exclude_Rows])

Table_Range

Array, Table, Range or Spill reference whose rows are to be averaged.

Exclude_Rows (Optional. Default no rows will be excluded)

Optional. If omitted all rows will be averaged.
A list (array or range) identifying the rows that are **not** to be averaged, given as one of:

- If Table_Range is not an array: One or more ranges intersecting rows in Table_Range. Multiple disjoint ranges must be enclosed in ( ) – for example (A1:A2,A4) results in rows 1,2 and 4.
- Alphabetic labels are used as lookup values in the first column of Table_Range to find the rows. These labels can be supplied in an array or a range.
- Relative row numbers are used directly.
  These row numbers can be supplied in an array or range.

  For example {2,4} gets the second and fourth row of the Table_Range.
  Negative numbers work from the bottom upwards so {-1, -2) gets the last two rows of the Table_Range.

  Zero acts as an include operator.
  {1,0,5} results in rows 1 through 5 inclusive
  {1,5} results in row 1 and row 5 only
  {5,0} results in row 5 and all rows after row 5
  {0,5} results in the first 5 rows

COUNTCOLS Function

Counts the values in each column of the Table_Range to produce a horizontal array of the counts of the columns. Either **all** information values (COUNTA) or only numbers (COUNT) can be counted.
Dynamically adjusts to the number of rows and columns in Table_Range:
use COUNTCOLS when you want non-floating totals for dynamic arrays.
COUNTCOLS is a multi-threaded, non-volatile array function.

COUNTCOLS Syntax

COUNTCOLS (Table_Range [, Exclude_Cols] [,All])

Table_Range

Array, Table, Range or Spill reference whose columns are to be counted.

Exclude_Cols (Optional. Default no columns will be excluded)

Optional. If omitted all columns will be counted.
A list (array or range) identifying the columns that are **not** to be counted, given as one of:

- If Table_Range is not an arry: One or more ranges within Table_Range. Multiple disjoint ranges must be enclosed in ( ) – for example (A1:B1,D1) results in columns 1,2 and 4.
- Alphabetic labels are used as lookup values in the first row of Table_Range to find the columns. These labels can be supplied in an array or a range.
- Relative column numbers are used directly.
  These column numbers can be supplied in an array or range.

  For example {2,4} gets the second and fourth column of the Table_Range.
  Negative numbers work from right to left so {-1, -2) gets the last two columns of the Table_Range.

  Zero acts as an include operator.
  {1,0,5} results in Columns 1 through 5 inclusive
  {1,5} results in Column 1 and Column 5 only
  {5,0} results in Column 5 and all Columns after Column 5
  {0,5} results in the first 5 Columns

All (Optional. Default TRUE)

Optional. Default True.
True counts cells containing any kind of information but not empty cells (COUNTA).
False only counts numbers (COUNT).

COUNTROWS Function

Counts the values in each row of the Table_Range to produce a vertical array of the counts of the rows.
Either **all** information values (COUNTA) or only numbers (COUNT) can be counted.
Dynamically adjusts to the number of rows and columns in Table_Range:
use COUNTROWS when you want non-floating totals for dynamic arrays.
COUNTROWS is a multi-threaded, non-volatile array function.

COUNTROWS Syntax

COUNTROWS (Table_Range [, Exclude_Cols] [, All])

Table_Range

Array, Table, Range or Spill reference whose rows are to be counted.

Exclude_Rows (Optional. Default no rows will be excluded)

Optional. If omitted all rows will be counted.
A list (array or range) identifying the rows that are **not** to be counted, given as one of:

- If Tanle_range is not an array: One or more ranges within Table_Range. Multiple disjoint ranges must be enclosed in ( ) – for example (A1:A2,A4) results in rows 1,2 and 4.
- Alphabetic labels are used as lookup values in the first column of Table_Range to find the rows. These labels can be supplied in an array or a range.
- Relative row numbers are used directly.
  These row numbers can be supplied in an array or range.

  For example {2,4} gets the second and fourth row of the Table_Range.
  Negative numbers work from the bottom upwards so {-1, -2) gets the last two rows of the Table_Range.

  Zero acts as an include operator.
  {1,0,5} results in rows 1 through 5 inclusive
  {1,5} results in row 1 and row 5 only
  {5,0} results in row 5 and all rows after row 5
  {0,5} results in the first 5 rows

All (Optional. Default TRUE)

Optional. Default True.
True counts cells containing any kind of information but not empty cells (COUNTA).
False only counts numbers (COUNT).

CORKSCREW Function

The CORKSCREW function provides a method of doing simple roll-forward calculations with Dynamic Arrays without using circular references and Excel iterative calculations.

These calculations start with an opening balance, add positive and negative flows to arrive at a closing balance, which then rolls forward to become the opening balance for the next column. The flows can be independent of or dependent on the opening balance. When dependent on the opening balance (for example an interest calculation) only simple arithmetic operations are allowed (multiply, divide, add, subtract, power).

Note: *In Dynamic Array Excel corkscrew calculations can be done using ordinary (non-spilling) references. Corkscrew calculations involving dynamic array references create a circular reference that can be calculated using iteration.*

CORKSCREW Syntax

CORKSCREW (Opening_Values, Op_Flow1, Op_Flow2 … [, Op_Flow12])

Opening _Values

One or more opening values to use as the starting Opening Balance. Can be a range reference, constant or array.

Op_Flow1

Each Flow parameter can refer to a range, array or spill reference. By default, all the Flow parameters are added to the opening balance to create the closing balance for this column, which then becomes the opening balance for the next column.

The Op_Flow parameter can also be a single string arithmetic operator (“*”, “/” , “+”, “-“, “^”) that defines an operation on the opening balance and the succeeding Flow.

Flows which are independent of the opening balance will usually be dynamic array references to calculations elsewhere in the workbook.

Flows which are in any way dependent on the opening balance in any column must be created using the method shown above.

CORKSCREW example

Two rows in B2:G3 give Sales and Costs derived from elsewhere in the model.

The initial opening balance is in A8 and the interest rate is in A9 (note that the interest rate could be a row that varies by year).

The CORKSCREW formula is **= CORKSCREW (A8,"*", A9,C2:G2, "-" , C3:G3)**

The "*" parameter causes the next parameter (A9=5%) to be used to multiply the opening balance by 5%.

The next parameter C2:G2 has no preceding operator so Sales will be added to the opening balance,

The next pair of parameters "-", C3:G3 subtracts Costs.

The closing balance is calculated by summing the opening balance and the intervening rows, and is used as the opening balance for the next period,

	A	B	C	D	E	F	G
1							
2		Sales	220.00	215.00	202.00	190.00	175.00
3		Costs	204.6	204.25	187.86	174.8	162.75
4							
5		**=CORKSCREW(A8,"*",A9,C2:G2,"-",C3:G3)**					
6							
7			**2019**	**2020**	**2021**	**2022**	**2023**
8	100	**Opening Balance**	100.00	120.40	137.17	158.17	181.28
9	5%	**Interest**	5.00	6.02	6.86	7.91	9.06
10		**Sales**	220.00	215.00	202.00	190.00	175.00
11		**Costs**	-204.60	-204.25	-187.86	-174.80	-162.75
12		**Closing Balance**	120.40	137.17	158.17	181.28	202.59

Array shaping functions

UNPIVOT Function

Transforms table columns from a Range, Array, Table or spill reference into rows. The UNPIVOT function is similar to Power Query unpivot, except that it is dynamically driven by the Excel calculation engine rather than by Power Query commands.
UNPIVOT allows for multiple header rows, including or excluding unpivot and other columns and providing labels for the attribute and value columns.
UNPIVOT is a multi-threaded, non-volatile array function.

UNPIVOT Syntax

UNPIVOT (Table_Range, [UnPiv_Columns], [Other_Columns], [Attrib_Names], [Value_Name] [, HasHeader])

Table_Range

Array, Table, Range or Spill reference to unpivot

UnPiv_Columns (Optional)

Optional. A one-dimensional array or range specifying which columns to unpivot.
Either UnPiv_Columns or Other_Columns or both should be specified.
*If Other_Columns is **not** specified all other columns will **not** be unpivoted.*
A list (array or range) identifying the columns that are to be unpivoted, given as one of:

- If Table_Range is not an array: One or more ranges within Table_Range. Multiple disjoint ranges must be enclosed in ( ) – for example (A1:B1,D1) results in columns 1,2 and 4.
- Alphabetic labels are used as lookup values in the first row of Table_Range to find the columns. These labels can be supplied in an array or a range.
- Relative column numbers are used directly.
  These column numbers can be supplied in an array or range.

  For example {2,4} gets the second and fourth column of the Table_Range.
  Negative numbers work from right to left so {-1, -2) gets the last two columns of the Table_Range.

  Zero acts as an include operator.
  {1,0,5} results in Columns 1 through 5 inclusive
  {1,5} results in Column 1 and Column 5 only
  {5,0} results in Column 5 and all Columns after Column 5
  {0,5} results in the first 5 Columns

Other_Columns (Optional)

Optional. A one-dimensional array or range specifying which columns **not** to unpivot.
Either UnPiv_Columns or Other_Columns or both should be specified.
*If UnPiv_Columns is **not** specified all other columns will be unpivoted.*
A list (array or range) identifying the columns **not** to be unpivoted, given as one of:

- If Table_range is not an array: One or more ranges within Table_Range. Multiple disjoint ranges must be enclosed in ( ) – for example (A1:B1,D1) results in columns 1,2 and 4.
- Alphabetic labels are used as lookup values in the first row of Table_Range to find the columns. These labels can be supplied in an array or a range.
- Relative column numbers are used directly.
  These column numbers can be supplied in an array or range.
  For example {2,4} gets the second and fourth column of the Table_Range.
  Negative numbers work from right to left so {-1,-2) gets the last two columns of the Table_Range.

Attrib_Names (Optional, default "Attribute")

Optional. Default "Attribute". A range or array or constant of the names to give to the columns of unpivoted column labels. (If no alphabetic column labels are found the column will be created using Column1, Column2, …)

Value_Name (Optional, default "Value")

Optional. Default "Value". The name to give to the column of unpivoted values.

HasHeader (Optional, indicates the number of header rows in Table_range)

Optional. Default -1: automatic detection of header
An integer value specifying if Table_Range has a header row:

- -1 If the first column of the first row of the UnPiv columns is alphabetic the first row is assumed to be a header row, else not.
- 0 Table_Range does not have a header row
- 1 Table_range has a single header row.
- N Table_range has N header rows.

UNPIVOT Example

(This example uses sample data from"M is for (DATA) Monkey" by Ken Puls and Miguel Escobar, ISBN 978-1-61547-034-1)

	A	B	C	D	E	F	G	H	I	J
1										
2		Sales in Units								
3										
4	Sales Category	1/1/2014	1/2/2014	1/3/2014	1/4/2014	1/5/2014	1/6/2014	1/7/2014	Total	1/8/20
5	Beer	103	243	101	137	103	185	111	983	34
6	Wine	175	223	138	57	66	199	83	941	86
7	Cider						78	92	170	47
8	Liquor	162	207	103	179	150	147	180	1,128	23
9	Total	440	673	342	373	319	609	466	3,222	

We want to unpivot this data, but ignore the Total column and include the extra column J the user has added after the Total column.

=UNPIVOT(Data!A4:J8,(Data!B4:H4,Data!J4),"Sales Category","Date","Units")

Data!A4:J8 gives the data we want to unpivot and ignores the Total Row.

(Data!B4:H4,Data!J4) gives the columns we want to unpivot and ignores the Total column.

"Sales Category" gives the column NOT to unpivot

"Date" and "Units" give labels for the attribute and Value.

The result of the UNPIVOT formula is

Sales Category	Date	Units
Beer	1/1/2014	103
Beer	1/2/2014	243
Beer	1/3/2014	101
Beer	1/4/2014	137
Beer	1/5/2014	103
Beer	1/6/2014	185
Beer	1/7/2014	111
Beer	1/8/2014	34
Wine	1/1/2014	175
Wine	1/2/2014	223
Wine	1/3/2014	138
Wine	1/4/2014	57
Wine	1/5/2014	66
Wine	1/6/2014	199
Wine	1/7/2014	83
Wine	1/8/2014	86
Cider	1/6/2014	78
Cider	1/7/2014	92
Cider	1/8/2014	47
Liquor	1/1/2014	162
Liquor	1/2/2014	207
Liquor	1/3/2014	103
Liquor	1/4/2014	179
Liquor	1/5/2014	150
Liquor	1/6/2014	147
Liquor	1/7/2014	180
Liquor	1/8/2014	23

SLICES Function

Returns an array composed of one or more slices of columns and rows from a Range or Array or Spill reference. Values are returned from the intersections of the given rows and columns.
If the rows and columns define a single contiguous area then SLICES will return a reference, otherwise SLICES will return an array with the non-contiguous areas stacked together into a single array.
Slices can use alphabetic labels to **lookup both rows and columns** and will return **all** the results found that match both conditions.
SLICES is a multi-threaded, non-volatile array function.

SLICES Syntax

SLICES (Range_Array, [SliceRows], [SliceColumns])

Range_Array

Array, Table, Range or Spill reference to take the slices from.

SliceRows (Optional, default 0)

Parameter giving the rows to return. A value of zero (default if SliceRows is omitted) means all the rows.
Rows can be derived from SliceRows in 3 different ways:

1. If Range_Array is a Range then SliceRows can be one or more ranges within Range_Array. Multiple disjoint ranges must be enclosed in ( )
   – for example (A1:A2,A4) results in rows 1,2 and 4.
2. Alphabetic labels are used as lookup values in the first column of Range_Array to find the rows. These labels can be supplied in an array or a non-interseting range.
3. Relative row numbers are used directly.
   These row numbers can be supplied in an array or non-intersecting range.

   For example {2,4} gets the second and fourth row of the Range_Array.
   Negative numbers work from the bottom upwards so {-1,-2} gets the last two rows of the Range_Array.

   Zero acts as an include operator.
   {1,0,5} results in rows 1 through 5 inclusive
   {2,0,-1} results in row 2 through to the second-to-last row.
   {1,5} results in row 1 and row 5 only
   {5,0} results in row 5 and all rows after row 5
   {0,5} results in the first 5 rows

SliceColumns (Optional, default 0)

Parameter giving the columns to return. A value of zero (default if SliceColumns is omitted) means all the columns.
Columns can be derived from SliceColumns in 3 different ways:

1. If Range_Array is a Range then SliceColumns can be one or more ranges within Range_Array. Multiple disjoint ranges must be enclosed in ( ) – for example (A1:B1,D1) results in columns 1,2 and 4.
2. Alphabetic labels are used as lookup values in the first row of Range_Array to find the columns. These labels can be supplied in an array or a range.
3. Relative column numbers are used directly. These column numbers can be supplied in an array or range. For example {2,4} gets the second and fourth column of the Range_Array. Negative numbers work from right to left so {-1, -2} gets the last two columns of the Range_Array.
   Zero acts as an include operator for columns in the same way as for rows.

SLICES Examples:

SLICES can use alphabetic labels as row and column lookups.

In this example SLICES looks in the Monthly table to find all rows that have a row label of “Pear” and the columns that have a header label of “Month” and “Revenue” and then returns an array of all the data from the intersections of these rows and columns.

=SLICES(Monthly[#All],"Pear",{"Month","Revenue"})

You could also use these formulas to get the same results.

- =SLICES(Monthly[#All],"Pear",(B4,E4)) – range reference to the column labels
- =SLICES(Monthly[#All],"Pear",{2,5}) - array of column numbers
- =SLICES(Monthly[#All],{3,7,11,15,19},{2,5}) - arrays of row and column numbers
- =SLICES(Monthly[#All],"Pear",(B3:B6,E3:E7)) - using ranges that intersect the header labels

Table Monthly **=SLICES(Monthly[#All],"Pear",{"Month","Revenue"})**

Fruit	Month	Quantity	Weight	Revenue
Apple	Jan	451	23452	135.3
Pear	Jan	383	16469	114.9
Orange	Jan	399	20748	199.5
Peach	Jan	424	22048	127.2
Apple	Feb	361	18772	144.4
Pear	Feb	428	18404	128.4
Orange	Feb	186	9672	55.8
Peach	Feb	443	23036	88.6
Apple	Mar	362	18824	108.6
Pear	Mar	402	17286	160.8
Orange	Mar	198	10296	59.4
Peach	Mar	281	14612	56.2
Apple	Apr	190	9880	57.0
Pear	Apr	100	4300	20
Orange	Apr	153	7956	30.6
Peach	Apr	473	24596	189.2
Apple	May	157	8164	62.8
Pear	May	495	21285	247.5
Orange	May	400	20800	160.0
Peach	May	395	20321	142.5

Jan	114.9
Feb	128.4
Mar	160.8
Apr	20
May	247.5

Given this range of data

	A	B	C
1	"1"	2	3
2	FALSE	5	6
3	7	8	9
4	Fred	10	11

SLICES(A1:C4,{2,3},{1,3}) will return this array

FALSE	6
7	9

SLICES(A1:C4,0,{2,3}) will return this array

2	3
5	6
8	9
10	11

To calculate Yield (total revenue /total sales) from floating totals use -1 to access the last row of the dynamic array:

=SLICES(K9#,-1,6)/SLICES(K9#,-1,4)

Filter:		Maryland	50
Yield:	55.00		

Invoice	Product	Region	Units	Price	Revenue
200	Digital Projector	Maryland	5	87.29	436.45
165	Digital Projector	Maryland	5	86.35	431.75
222	Charging Station	Maryland	5	51.76	258.80
40	Charging Station	Maryland	4	53.17	212.68
30	Charging Station	Maryland	4	51.48	205.92
106	Charging Station	Maryland	3	53.93	161.79
5	Home Automation Hub	Maryland	5	24.78	123.90
172	Home Automation Hub	Maryland	4	23.75	95.00
12	Charging Station	Maryland	1	53.81	53.81
			36		1,980.10

SPLICECELLS function

Replace up to 5 cells with values in an array.
SPLICECELLS is a multi-threaded, non-volatile array function.

SPLICECELLS Syntax

SPLICECELLS (Range_Array, Rows, Columns, Splice [, Rows2, Cols2, Splice2] [, Rows3, Cols3, Splice3] [, Rows4, Cols4, Splice4] [, Rows5, Cols5, Splice5])

Range_Array

Array, Table, Range or Spill reference be spliced.

Rows (Optional, default -1)

Optional: the row number in Range_Array at which the cell will be replaced, -n = the nth row counting upwards from the bottom, +n = the nth row counting downwards

Cols (Optional, default -1)

Optional: the column number in Range_Array at which the cell will be replaced, -n = the nth column counting leftwards from the right, +n = the nth column counting rightwards from the left

Splice (optional)

Optional.: the value to replace the cell found at Rows, Cols

SPLICEROWS function

Removes and/or replaces rows within a range, array or spill reference, after a specified row or at the beginning or end of the range/array.
SPLICEROWS is a multi-threaded, non-volatile array function.

SPLICEROWS Syntax

SPLICEROWS (Range_Array, [AfterRow], [RemoveRows], [Splice], [Fill])

Range_Array

Array, Table, Range or Spill reference be spliced.

AfterRow (Optional, default -1)

Optional: the row number in Range_Array after which rows will be inserted or deleted, -1 = at the end of Range_Array, 0 = at the start of Range_Array

RemoveRows (Optional, default 0)

Optional: the number of rows to be removed from the Range_Array, Default 0 = no rows, -1 = all rows after AfterRow

Splice (optional)

Optional.: the array, range or spill ref to be inserted into Range_Array after AfterRow

Fill (optional, default #N/A)

If Splice has fewer columns than Range_Array the Fill value will be used to fill the missing cells after insertion into Range_Array. Default is #N/A.

SPLICECOLS function

Removes and/or replaces columns within a range, array or spill reference, after a specified column or at the right or left of the range/array.
SPLICECOLS is a multi-threaded, non-volatile array function.

SPLICECOLS Syntax

SPLICECOLS (Range_Array, [AfterColumn], [RemoveColumns], [Splice], [Fill])

Range_Array

Array, Table, Range or Spill reference be spliced.

AfteColumn (Optional, default -1)

Optional: the column number in Range_Array after which columns will be inserted or deleted, -1 = to the right of Range_Array, 0 = to the left of Range_Array

RemoveColumns (Optional, default 0)

Optional: the number of columns to be removed from the Range_Array, Default 0 = no columns, -1 = all columns to the right of AfterColumn

Splice (optional)

Optional.: the array, range or spill ref to be inserted into Range_Array after AfterColumn

Fill (optional, default #N/A)

If Splice has fewer rows than Range_Array the Fill value will be used to fill the missing cells after insertion into Range_Array. Default is #N/A.

REPEAT function

Copies a range/array consisting of a rectangular block of cells one or more times vertically and/or horizontally by duplicating the block of cells.
REPLACE is a multi-threaded, non-volatile array function.

REPEAT Syntax

REPEAT (RangeArray, [VerticalRepeats] [,HorizontalRepeats])

RangeArray

Array, Table, Range or Spill reference to duplicate

VerticalRepeats (Optional, default=1)

The number of times to stack a copy of the input RangeArray vertically above one another.

HorizontalRepeats (Optional, default=1)

The number of times to stack a copy of the input RangeArray horizontally alongside one another.

REPEAT Examples:

Q1	Q2	Q3	Q4				
	=REPEAT(B18:E18,,2)						
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4

1	2					
3	4	=REPEAT(B9:C10,3,2)				
		1	2	1	2	
		3	4	3	4	
		1	2	1	2	
		3	4	3	4	
		1	2	1	2	
		3	4	3	4	

Whereas the REPT function repeats a string of text by concatenating it to itself n times, the REPEAT function repeats a string of text by copying it to adjacent cells n times.

RESIZE Function

Produces an expanded or contracted array copy of an array, range, Table or spill reference. Expanded cells may be filled with a Fill value.
RESIZE is a multi-threaded, non-volatile array function.

RESIZE Syntax

RESIZE (InputArray [, NumRows] [, NumCols] [, FirstRow] [, FirstCol] [, Fill])

InputArray

Array, Table, Range or Spill reference to resize.

NumRows (Optional, default -1= all rows)

Optional: the number of rows in the output array. Can be smaller or larger than the input array. Default is all rows: output size is the number of rows in an array formula entered into multiple cells, or the number of input rows in a single-cell formula.

NumCols (Optional, default -1= all columns)

Optional: the number of columns in the output array. can be smaller or larger than the input array. Default is all rows: output size is the number of columns in an array formula entered into multiple cells, or the number of input columns in a single-cell formula.

FirstRow (Optional, default the first row in the InputArray)

Optional: default the first row in InputArray. The start row in InputArray for the resize

FirstCol (Optional, default the first column in the InputArray)

Optional: default the first column in InputArray. The start column in InputArray for the resize

Fill (Optional, default 0.0)

A cell or character or digit to be used to fill expanded cells in the output array, for example 0.0 or “”.

REVERSE.ARRAY Function

Reverses the rows or columns (or both) of an array.
REVERSE.ARRAY is a multi-threaded, non-volatile array function.

REVERSE.ARRAY Syntax

REVERSE.ARRAY (InputArray, ReverseRows, ReverseCols)

InputArray

An expression or range that returns an array of values.

ReverseRows (Optional, default TRUE)

TRUE to reverse the rows of the array so that the last row becomes the first row.

ReverseCols (Optional, default TRUE)

TRUE to reverse the columns of the array so that the last column becomes the first row.

REVERSE.ARRAY Examples

S27:U29 contains

1	2	3
4	5	6
7	8	9
10	11	12

= REVERSE.ARRAY(S27:U30,TRUE,FALSE) reverses the rows but not the columns and returns

10	11	12
7	8	9
4	5	6
1	2	3

=REVERSE.ARRAY(S27:U30,FALSE,TRUE) reverses the columns but not the rows and returns

3	2	1
6	5	4
9	8	7
12	11	10

=REVERSE.ARRAY(S27:U30,TRUE,TRUE) reverses both rows and columns and returns

12	11	10
9	8	7
6	5	4
3	2	1

By comparison =TRANSPOSE(S27:U30) returns

1	4	7	10
2	5	8	11
3	6	9	12

VSTACK & VSTACKF Functions

VSTACKF is an alias of FastExcel VSTACK, created to enable formulas to distinguish between native Excel VSTACK and FastExcel VSTACK. Creates a single array from multiple arrays or ranges by appending up to 29 Range /Array arguments below one another row wise. Missing cells in the output array can be filled with Fill characters, for example 0.0 or "".
VSTACK is a multi-threaded, non-volatile array function.

VSTACK & VSTACKF Syntax

VSTACK ([Fill], Arg1, Arg2, … , Arg28)

Fill (Optional, default #N/A)

A cell or character or digit to be used to fill unused cells in the output array, for example 0.0 or “”.
When the Arrays or ranges to be appended do not have equally sized dimensions the gaps will be filled with the Pad character. Use zero if you want to append differently-sized numeric arrays and pass the output to an aggregating function such as SUM() or a single-cell array formula.

Note: *A non-array argument given for the first argument to VSTACK will be treated as a fill value. Preface the non-array argument with a comma to make it vthe first VSTACK item.*

Arg1 to Arg28

Ranges or arrays to be appended. Each Range/Array is placed in the output array starting in column 1 of the next available output row.
The ranges to be appended can be on different sheets.
If an Argument range/array is a constant or resolves to a single value it will be propagated across the entire row.

With the introduction of the native Excel VSTACK function VSTACKF allows you to choose either the native or FastExcel VSTACK implementation.

VSTACK Examples

A2:C2 contains 1,2,3
B4:D4 contains 4,5,6
{=VSTACK(A2:C2,B4:D4)} array entered into O20:Q21 returns

1	2	3
4	5	6

(A2:C2 cannot be a FILL character so is treated as Arg1)
{=VSTACK(0,A2:C2,{100;200;300},B4:D4)} array entered into C23:F28 returns

1	2	3	#N/A
100	0	0	#N/A
200	0	0	#N/A
300	0	0	#N/A
4	5	6	#N/A
#N/A	#N/A	#N/A	#N/A

The column dimension of the output array is the maximum number of columns in any of the arguments.
The row dimension of the output array is the sum of the number of rows in each argument. So the output array is 5 rows deep by 3 columns, which Excel has expanded with #N/A to the number of rows (6) and columns (4) the array formula was entered into.
The values from each argument are placed in turn into the output array, appending row-wise to the bottom.

For the arguments that have less than 3 columns the missing rows are filled with the fill character (zero in this case). Because the formula was CSE array-entered into 6 rows by 4 columns Excel has added #N/A in the 6th row and 4th column.
Entering =SUM(VSTACK(A2:C2,{100;200;300},B4:D4)) (NOT as an array formula) returns #N/A, because the array returned from ROW.ARRAY contains #N/A.
Entering =SUM(VSTACK(0,A2:C2,{100;200;300},B4:D4)) (NOT as an array formula) returns 621, which is the sum of the values in the output array (the #N/A in the array has been replaced by zero).
Entering =AVERAGE(VSTACK(0,A2:C2,{100;200;300},B4:D4)) (NOT an array formula) returns 41.4 which is 621 divided by the number of cells in the output array (15).

HSTACK & HSTACKF Function

HSTACKF is an alias of HSTACK, created to enable formulas to distinguish between native Excel HSTACK and FastExcel HSTACK.. Creates a single array from multiple arrays or ranges by appending up to 29 Range /Array arguments alongside one another column wise. Missing cells in the output array can be filled with Fill characters, for example 0.0 or "".
HSTACK is a multi-threaded, non-volatile array function.

HSTACK & HSTACKF Syntax

HSTACK ([Fill], Arg1, Arg2, … , Arg28) or HSTACKF ([Fill], Arg1, Arg2, … , Arg28)

Fill (Optional, default #N/A)

A cell or character or digit to be used to fill unused cells in the output array, for example 0 or "".
When the Arrays or ranges to be appended do not have equally sized dimensions the gaps will be filled with the Fillcharacter. Use zero if you want to append differently-sized numeric arrays and pass the output to an aggregating function such as SUM() or a single-cell array formula.

Note: *A non-array argument given for the first argument to HSTACK will be treated as a fill value. Preface the non-array argument with a comma to make it vthe first HSTACK item.*

Arg1 to Arg28

Ranges or arrays to be appended. Each Range/Array is placed in the output array starting in row 1 of the next available output column.
The ranges to be appended can be on different sheets.
If an Argument range/array is a constant or resolves to a single value it will be propagated down the entire column.

With the introduction of the native Excel HSTACK function HSTACKF allows you to choose either the native or FastExcel HSTACK implementation.

HSTACK Examples

A2:C2 contains 1,2,3
B4:D4 contains 4,5,6
{=HSTACK(A2:C2,B4:D4)} CSE array entered into C20:H20 returns

1	2	3	4	5	6

(A2:C2 cannot be a PAD character so is treated as Arg1)
{=HSTACK(0,A2:C2,{100;200;300},B4:D4)} array entered into C7:J10 returns

1	2	3	100	4	5	6	#N/A
0	0	0	200	0	0	0	#N/A
0	0	0	300	0	0	0	#N/A
#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A

The row dimension of the output array is the maximum number of rows in any of the arguments. The column dimension of the output array is the sum of the number of columns in each argument. So the output array is 3 rows deep by 7 columns.
The values from each argument are placed in turn into the output array, appending column-wise to the right.
For the arguments that have less than 3 rows the missing rows are filled with the fill character (zero in this case). Because the formula was array-entered into 4 rows by 8 columns Excel has added #N/A in the 4th row and 8th column.
Entering =SUM(HSTACK(0,A2:C2,{100;200;300},B4:D4)) (NOT as an array formula) returns 621, which is the sum of the values in the output array.
Entering =AVERAGE(HSTACK(0,A2:C2,{100;200;300},B4:D4)) (NOT an array formula) returns 29.57 which is 621 divided by the number of cells in the output array (21).

HDR.VSTACK Function

Matches cells in the first header row of the array/range to header row cells from multiple arrays or ranges to create an output array with a single header row. Up to 29 Range /Array arguments can be appended below one another row wise.
Each column with a matching header is stacked below it's matching column.
Columns in subsequent arrays/ranges where column headers do not match the first array/range column header are filled with Fill characters, for example #N/A (default) or 0.0 or "".
HDR.VSTACK is a multi-threaded, non-volatile array function.

VSTACK Syntax

HDR.VSTACK ([Fill], Arg1, Arg2, … , Arg29)

Fill (Optional, default #N/A)

A cell or character or digit to be used to fill unused cells in the output array, for example 0.0 or "".
When the Arrays or ranges to be appended do not have matching column headers the gaps will be filled with the Fill character. Use zero if you want to append differently-sized numeric arrays or arrays with some different header cells and pass the output to an aggregating function such as SUM() or a single-cell array formula.

Note: *A non-array argument given for the first argument to HDR.VSTACK will be treated as a fill value. Preface the non-array argument with a comma to make it the first HDR.VSTACK item.*

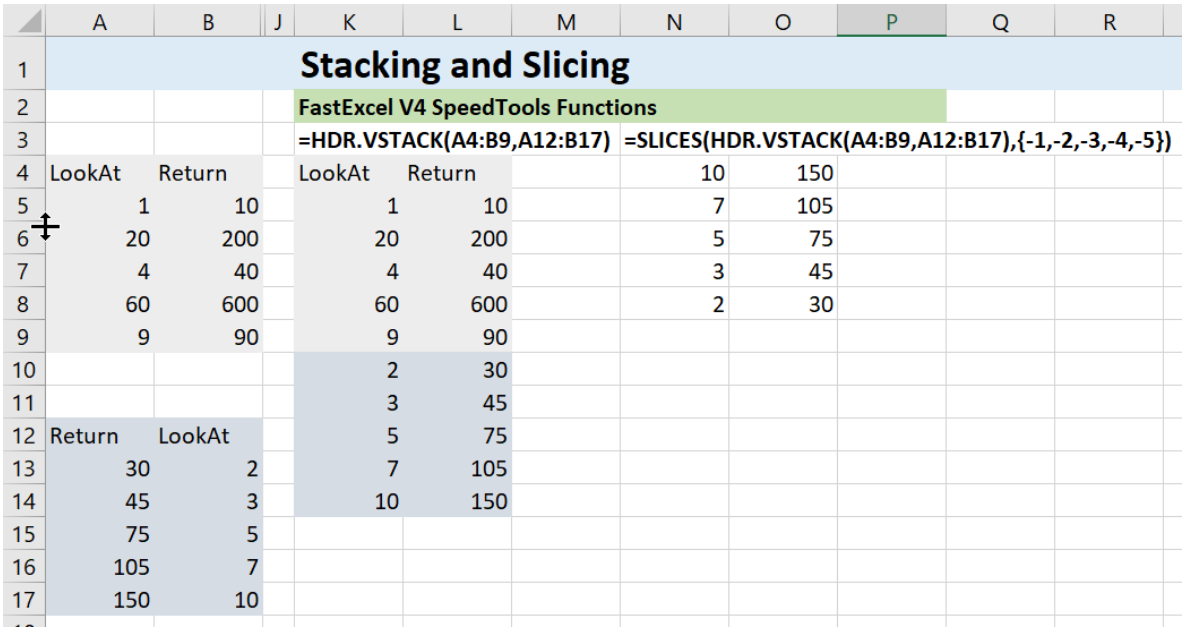
Arg1 to Arg28

Ranges or arrays to be appended.
Where cell values in the first row of each Range/Array match the cell values in the first row of the first range/array the columns are placed in the output array in that matched column. The sequence of columns can be different in each array/range.
Only the first row (header row) in the first range/array is copied into the output array. The header rows in subsequent arrays/ranges are **not** copied.
The ranges to be appended can be on different sheets.
If an Argument range/array is a constant or resolves to a single value it will be propagated across the entire row.

Stacking and Slicing Examples

The first formula =HDR.VSTACK(A4:B9,A12:B17) stacks the 2 ranges on top of one another, but re-arranges the columns from the second range to match the columns from the first range by using the column headers.

The second formula takes slices out of the array returned by the HDR.VSTACK but uses an array of relative row numbers working upwards from the bottom of the array **{-1,-2,-3,-4,-5}**



	A	B	J	K	L	M	N	O	P	Q	R
1				**Stacking and Slicing**							
2				**FastExcel V4 SpeedTools Functions**							
3				**=HDR.VSTACK(A4:B9,A12:B17)**			**=SLICES(HDR.VSTACK(A4:B9,A12:B17),{-1,-2,-3,-4,-5})**				
4	LookAt	Return		LookAt	Return		10	150			
5	1	10		1	10		7	105			
6	20	200		20	200		5	75			
7	4	40		4	40		3	45			
8	60	600		60	600		2	30			
9	9	90		9	90						
10				2	30						
11				3	45						
12	Return	LookAt		5	75						
13	30	2		7	105						
14	45	3		10	150						
15	75	5									
16	105	7									
17	150	10									

HDR.HSTACK Function

Matches cells in the first header column of the array/range to header column cells from multiple arrays or ranges to create an output array with a single header column. Up to 29 Range /Array arguments can be appended alongside one another column wise.
Each row with a matching header is stacked alongside it's matching row.
Rows in subsequent arrays/ranges where row headers do not match the first array/range row header are filled with Fill characters, for example #N/A (default) or 0.0 or "".
HDR.HSTACK is a multi-threaded, non-volatile array function.

HDE.HSTACK Syntax

HDR.HSTACK ([Fill], Arg1, Arg2, … , Arg29)

Fill (Optional, default #N/A)

A cell or character or digit to be used to fill unused cells in the output array, for example 0.0 or "".
When the Arrays or ranges to be appended do not have matching row headers the gaps will be filled with the Fill character. Use zero if you want to append differently-sized numeric arrays or arrays with some different header cells and pass the output to an aggregating function such as SUM() or a single-cell array formula.

Note: *A non-array argument given for the first argument to HDR.HSTACK will be treated as a fill value. Preface the non-array argument with a comma to make it the first HDR.HSTACK item.*

Arg1 to Arg28

Ranges or arrays to be appended.
Where cell values in the first column of each Range/Array match the cell values in the first column of the first range/array the rows are placed in the output array in that matched row. The sequence of rows can be different in each array/range.
Only the first column (header column) in the first range/array is copied into the output array. The header columns in subsequent arrays/ranges are **not** copied.
The ranges to be appended can be on different sheets.
If an Argument range/array is a constant or resolves to a single value it will be propagated down the entire column.

VECTOR Function

VECTOR returns a column or row of increasing or decreasing positive or negative integers
VECTOR is a multi-threaded, non-volatile array function.

VECTOR Syntax

VECTOR (StartValue, EndValue, Step, Row, Fill)

StartValue (Optional, default 1)

The first number. Can be positive or negative. If omitted the value 1 will be used

EndValue (Required)

The last number: can be positive or negative.

Step (Optional, default 1)

The increment between numbers. Can be positive or negative. If omitted the value 1 or -1 will be used. If the sign of step is not a valid increment for StartValue and EndValue its sign will be reversed.

Row (Optional, default TRUE)

If ROW is TRUE then a row of numbers will be generated. If FALSE a Column of numbers will be generated.

Fill (Optional, default "")

If VECTOR is CSE array entered into more cells than the number of integers requested, Excel would normally put #N/A into the excess cells. The Fill parameter provides the value to use instead of #N/A.

VECTOR Examples

=VECTOR(,6) returns a row containing 1 to 6
=VECTOR(1,6,1,FALSE) returns a column containing 1 to 6
=VECTOR(6,1,-1,FALSE) returns a column containing 6 to 1
=VECTOR(-6,-1,-1,FALSE) returns a column containing -6 to -1
=VECTOR(-3,2,-1,FALSE) returns a column containing -3 to +2
=VECTOR(-3,6,-2,FALSE) returns a column containing -3,-1,1,3,5

Join, Compare and Merge Functions

JOIN, MATCHES, ANTIJOIN, MISSMATCHES and COMPARE.LISTS provide efficient ways of dynamically merging, joining and comparing rows of data.

ANTIJOIN MISSMATCH

MISSMATCH is an alias of ANTIJOIN.

Finds the miss-matched rows (anti-join) in two ranges/arrays/tables/lists.

The matches can be done on one or more chosen columns.
Matches are not case-sensitive so “ID” matches with “id”.
Numbers are treated as text, so 42 matches with both “42” and 42.

You can specify which columns from the Left range or array to match with which columns from the Right range or array. Columns that are not used to match will still appear in the output.

ANTIJOIN/MISSMATCHES Syntax

ANTIJOIN/MISSMATCHES (Left, Right, Match_Type, Match_Cols, No_Match)

Left (Required)

The Left range/array/list/table.

Right (Required)

The Right range/array/list/table.

Match_Type (Optional: default = 3)

The type of output match requested:

- 1 = All rows from Left that have no match in Right
- 2 = All rows from Right that have no match in Left
- 3 = All rows from Left that have no match in Right, and (below) all rows from Right that have no match in Left

Match_Type is optional; if omitted or empty, the value 3 will be used.

Match_Cols (Optional: defaullt = -1)

The columns to match on:

- -1 = match columns on column headers. The first row of Left and of Right is assumed to contain column headers. The column headers do not have to be in the same sequence. Any columns whose headers that do not match are ignored by the matching process but are shown in the output.
- 0 = match on all columns in sequence
- Array or range = Column numbers or header names to match on. If a single row then these column numbers/header names will be use for both Left and Right.
  If 2 rows then the first row gives the column numbers/header names for left and the second row gives the corresponding column numbers/header names from Right.

  For example {1,3;2;4} matches column 1 in Left with column 2 from Right and column 3 from Left with column 4 from Right, and {“Brand”;”Alternate Brand”} would look for a column header of “Brand” in Left and “Alternate Brand” in Right.

Skip_LeftCols & Skip_RightCols (Optional: default=0)

Optional: default = 0. Columns from Left or Right to omit from the output. 0 means show all columns. A single-row range or array giving column numbers or header labels.

No_Match (Optional: default= #Null)

Value to use for rows that do not match.

JOIN MATCHES

MATCHES is an alias of JOIN
Finds the matched rows (join) in two ranges/arrays/tables/lists and joins them.
JOIN/MATCHES can handle one-to-one, one-to-many, and many-to-many joins.

The matches can be done on one or more chosen columns.
Matches are not case-sensitive so “ID” matches with “id”.
Numbers are treated as text, so 42 matches with both “42” and 42.
You can specify which columns from the Left range or array to match with which columns from the Right range or array. Columns that are not used to match will still appear in the output.

JOIN/MATCHES Syntax

JOIN/MATCHES (Left, Right, Match_Type, Match_Cols, Skip_LeftCols, Skip_RightCols, No_Match)

Left (Required)

The Left range/array/list/table.

Right (Required)

The Right range/array/list/table.

Match_Type (Optional: default = 4)

The type of output match requested:

- 4 = All rows from Left with any matching rows from Right
- 5 = Only matching rows from Left with their matching rows from Right
- 6 = All rows from Right with any matching rows from Left
- 7 = Only matching rows from Right with their matching rows from Left

Match_Type is optional; if omitted or empty, the value 4 will be used.

Match_Cols (Optional: defaullt = -1)

The columns to match on:

- -1 = match columns on column headers. The first row of Left and of Right is assumed to contain column headers. The column headers do not have to be in the same sequence. Any columns whose headers that do not match are ignored by the matching process but are shown in the output.
- 0 = match on all columns in sequence
- Array or range = Column numbers or header names to match on.
  If a single row then these column numbers/header names will be use for both Left and Right. If 2 rows then the first row gives the column numbers/header names for left and the second row gives the corresponding column numbers/header names from Right.

  For example {1,3;2;4} matches column 1 in Left with column 2 from Right and column 3 from Left with column 4 from Right, and {“Brand”;”Alternate Brand”} would look for a column header of “Brand” in Left and “Alternate Brand” in Right.

Skip_LeftCols & Skip_RightCols (Optional: default=0)

Optional: default = 0. Columns from Left or Right to omit from the output. 0 means show all columns. A single-row range or array giving column numbers or header labels.

No_Match (Optional: default= #Null)

Value to use for rows that do not match.

JOIN, ANTIJOIN, MISSMATCHES and MATCHES Examples

(This example uses **modified** sample data from"M is for (DATA) Monkey" by Ken Puls and Miguel Escobar, ISBN 978-1-61547-034-1)

The Inventory Items table

SKU Numbe	Brand	Type	Unit	Pack Quantit	Sales Price	Unit Cos	Margin
510007	Budweiser	Lager	Cans	15	29.5	24.4	5.1
510010	Canadian	Lager	Cans	6	12	9.95	2.05
510014	Canterbury	Ale	Cans	6	11.5	9.5	2
510019	Corona Extra	Lager	Bottles	6	13.5	11.25	2.25
510022	Corona Grande	Lager	Bottles	1	4.5	3.7	0.8
510032	Granville Islan	Ale	Bottles	6	13	10.95	2.05
510037	Guinness	Stout	Cans	4	13	10.99	2.01
510038	Heineken	Lager	Bottles	6	14.5	11.95	2.55
510046	Kokanee	Lager	Tall Cans	6	15.5	13.05	2.45
510057	Miller	Lager	Bottles	12	24	20.2	3.8
510059	OK Springs	Lager	Bottles	6	13	10.99	2.01
510065	OK Springs	Ale	Bottles	6	13	10.99	2.01
510077	Ghost Ship	Ale	Bottles	6	14	11.15	2.85

The sales transactions table

Date	SKU Numbe	Brand	Sales Quantit
2014-03-12	510007	Budweiser	64
2014-03-12	510010	Canadian	45
2014-03-12	510014	Canterbury	62
2014-03-12	510019	Corona Extra	64
2014-03-14	510019	Corona Extra	24
2014-03-12	510021	Corona Grande	38
2014-03-14	510021	Corona Grande	31
2014-03-12	510032	Granville Island	24
2014-03-13	510032	Granville Island	30
2014-03-14	510033	Granville Island	48
2014-03-12	510037	Guinness	73
2014-03-13	510037	Guinness	76
2014-03-14	510037	Guinness	74
2014-03-12	510038	Heineken	30
2014-03-13	510038	Heineken	30
2014-03-14	510039	Heineken	44
2014-03-12	510046	Kokanee	57
2014-03-12	510057	Miller	43
2014-03-12	510059	OK Springs	76
2014-03-12	510065	OK Springs	78

There are potential problems with this data: the sales table contains SKUs that do not exist in the inventory table, and the inventory table has SKUs that do not exist in the Sales table. We can use MISSMATCHES to find the miss-matches between the two tables.

=MISSMATCHES(K7:N27,A7:H20,,,"**") using the default match type of 3 and a No_Match of "**" produces

41710	510021	Corona Grande	38	**	**	**	**	**	**	**	**
41712	510033	Granville Island	48	**	**	**	**	**	**	**	**
41712	510039	Heineken	44	**	**	**	**	**	**	**	**
**	**	**	**	510022	Corona Grande	Lager	Bottles	1	4.5	3.7	0.8
**	**	**	**	510077	Ghost Ship	Ale	Bottles	6	14	11.15	2.85

Using match type 1 only shows the miss-matches in the left transactions table:

41710	510021	Corona Grande	38
41712	510033	Granville Island	48
41712	510039	Heineken	44

And match type 2 only shows the miss-matches from the right inventory table:

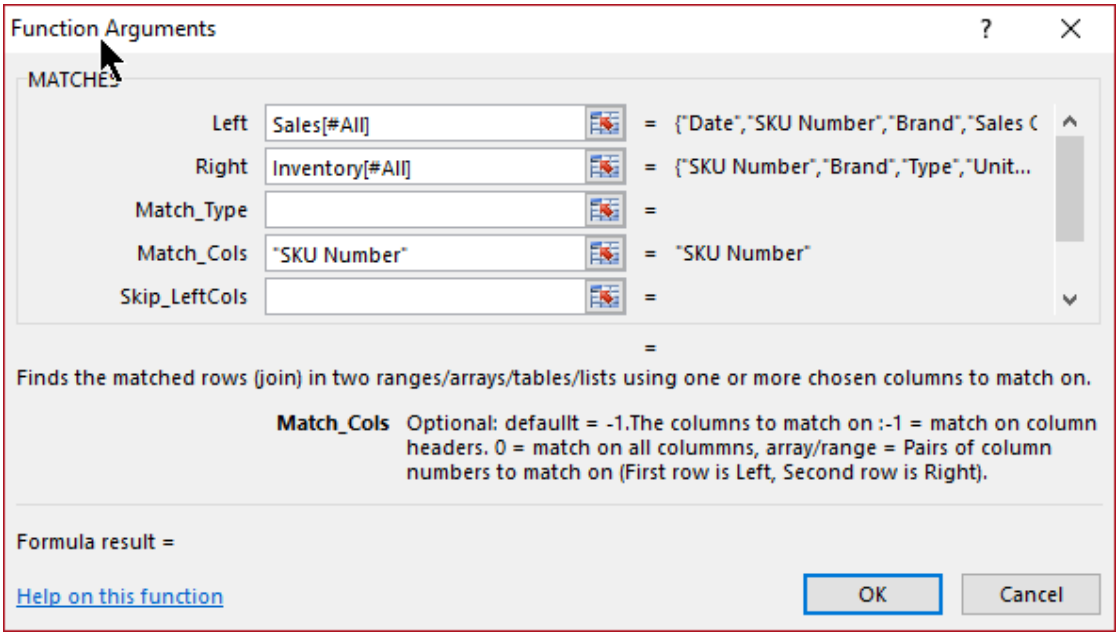
510022	Corona Grande	Lager	Bottles	1	4.5	3.7	0.8
510077	Ghost Ship	Ale	Bottles	6	14	11.15	2.85

To see both the matches and miss-matches for the left transaction table use this formula;

=MATCHES(Sales[#All],Inventory[#All])

But the SKU Number and Brand appear twice in the output (once from the left table and once from the right table). And often you need to specify which columns to use for the match, rather than using the default of matching on column headers.

Use the function wizard to see expanded details for the arguments and to call up help for the function.



=MATCHES(Sales[#All],Inventory[#All],,"SKU Number",,{"Brand","Sku number"})

Date	SKU Number	Brand	Sales Quantity	Type	Unit	Pack Quar	Sales Pric	Unit Cost	Margin
12 March 2014	510007	Budweise	64	Lager	Cans	15	29.5	24.4	5.1
12 March 2014	510010	Canadian	45	Lager	Cans	6	12	9.95	2.05
12 March 2014	510014	Canterbur	62	Ale	Cans	6	11.5	9.5	2
12 March 2014	510019	Corona Ex	64	Lager	Bottles	6	13.5	11.25	2.25
14 March 2014	510019	Corona Ex	24	Lager	Bottles	6	13.5	11.25	2.25
12 March 2014	510021	Corona Gr	38	#NULL!	#NULL!	#NULL!	#NULL!	#NULL!	#NULL!
14 March 2014	510021	Corona Gr	31	#NULL!	#NULL!	#NULL!	#NULL!	#NULL!	#NULL!
12 March 2014	510032	Granville I	24	Ale	Bottles	6	13	10.95	2.05
13 March 2014	510032	Granville I	30	Ale	Bottles	6	13	10.95	2.05
14 March 2014	510033	Granville I	48	#NULL!	#NULL!	#NULL!	#NULL!	#NULL!	#NULL!
12 March 2014	510037	Guinness	73	Stout	Cans	4	13	10.99	2.01
13 March 2014	510037	Guinness	76	Stout	Cans	4	13	10.99	2.01
14 March 2014	510037	Guinness	74	Stout	Cans	4	13	10.99	2.01
12 March 2014	510038	Heineken	30	Lager	Bottles	6	14.5	11.95	2.55
13 March 2014	510038	Heineken	30	Lager	Bottles	6	14.5	11.95	2.55
14 March 2014	510039	Heineken	44	#NULL!	#NULL!	#NULL!	#NULL!	#NULL!	#NULL!
12 March 2014	510046	Kokanee	57	Lager	Tall Cans	6	15.5	13.05	2.45
12 March 2014	510057	Miller	43	Lager	Bottles	12	24	20.2	3.8
12 March 2014	510059	OK Spring	76	Lager	Bottles	6	13	10.99	2.01
12 March 2014	510065	OK Spring	78	Ale	Bottles	6	13	10.99	2.01

COMPARE.LISTS Function

This function does a dynamic compare of two lists, a LookFor list and a LookIn list. The function is designed to efficiently dynamically compare two lists of items and find either the items that do match or the items that don't match,

COMPARE.LISTS can also be used as a condition in FILTER functions.

COMPARE.LISTS is capable of comparing 2 lists, each containing more half a million items, in a third of a second.

COMPARE.LISTS can return:

- True/False or ""/"**" for each item in the LookFor list showing whether or not it can be found the LookIn list.
- A count of the items in LookFor NOT FOUND in LookIn
- A count of the items in LookFor FOUND in LookIn
- A list of the items in LookFor NOT FOUND in LookIn
- A list of the items in LookFor FOUND in LookIn

COMPARE.LISTS is a multi-threaded, non-volatile array function.
COMPARE.LISTS can handle whole-column references efficiently.

COMPARE.LISTS Syntax

COMPARE.LISTS (LookFor, LookIn, Output, Case_Sensitive)

The first two parameters are required.

LookFor (Required)

A single-column vertical range or array of data containing the items to be searched for in the LookIn list.

Empty cells in the LookFor list are ignored.

LookIn (Required)

A single-column vertical range or array of data containing the list of items to be searched for each item in the LookFor list.

Empty cells in the LookIn list are ignored.

If either of LookFor or LookIn refers to a single empty cell COMPARE.LISTS will return #Value.

Output (Optional: Default=1 if entered as a dynamic array formula, otherwise 3)

This option controls the type of output returned from COMPARE.LISTS.

Output=1: An array of True/False for each item in the LookFor list showing whether or not it can be found in the LookIn list.
When using Output=1 enter the COMPARE.LIST function as a multi-cell array formula alongside the LookFor list or as a dynamic array formula so that you get a corresponding True or False for each item in LookFor.

Output=2: An array of "" or "**" for each item in the LookFor list showing whether or not it can be found the LookIn list. "**" indicates that the item was not found.
When using Output=2 enter the COMPARE.LIST function as a multi-cell array formula alongside the LookFor list or as a dynamic array formula so that you get a "" or "**" for each item in LookFor.

Output=3: A Count and list of the items in LookFor that are NOT found in LookIn. The first output cell contains the count. To get only the count of non-matching items enter COMPARE.LISTS as a single-cell formula.

Output=4: A Count and list of the items in LookFor that ARE found in LookIn. The first output cell contains the count. To get only the count of matching items enter COMPARE.LISTS as a single-cell formula.

Output=5: A list of the items in LookFor that are NOT found in LookIn (No count).

Output=6: A list of the items in LookFor that ARE found in LookIn (No count).

Case_Sensitive (Optional: Default=False)

This option controls whether text will be compared case-sensitively or not. It can be True or False, and the default value is False so that by default lower-case text matches upper-case text.

COMPARE.LISTS Examples

These examples use the following LookIn and LookFor lists

	A	B	C
5	**LookIn**		**LookFor**
6	1		2
7	2		plug
8	3		fred
9	fred		9
10	FRED		24/09/1996
11	24/09/1996		23/09/1996
12	TRUE		FALSE
13	FALSE		0
14	#N/A		
15			#N/A
16	zap		

Note: although some of these cells appear empty A15, A16, C13, C16 are not empty,

- A8 is empty.
- A15 contains a single ‘ character
- A16 contains a single space character
- C13 contains a single space character
- C14 is empty
- C16 contains a single ‘ character

OUTPUT=1(Default): TRUE/FALSE

	A	B	C	D
5	**LookIn**		**LookFor**	**=COMPARE.LISTS($C$6:$C$17,$A$6:$A$17)**
6	1		2	TRUE
7	2		plug	FALSE
8			fred	TRUE
9	fred		9	FALSE
10	FRED		24/09/1996	TRUE
11	24/09/1996		23/09/1996	FALSE
12	TRUE		FALSE	TRUE
13	FALSE			TRUE
14	#N/A			
15			#N/A	TRUE
16				TRUE
17				

The results in cell D13 and D14 are blank because the empty cells in C13 and C17 have been ignored. Cells C13 and C16 show TRUE because the LookIn list contains cells with a single space character and a single ‘ character.

Using OUTPUT=1 you can filter the LookFor list for matches, mismatches or empty cells.

*OUTPUT=2: ** or space*

	A	B	C	E
5	**LookIn**		**LookFor**	**{=COMPARE.LISTS($C$6:$C$17,$A$6:$A$17,2)}**
6	1		2	
7	2		plug	**
8			fred	
9	fred		9	**
10	FRED		24/09/1996	
11	24/09/1996		23/09/1996	**
12	TRUE		FALSE	
13	FALSE			
14	#N/A			
15			#N/A	
16				
17				

Using OUTPUT=2 you get ** for each mismatch: this is easier to see than visually scanning for FALSE.

OUTPUT = 3 and 4, single-cell non-array formula:

Count Missmatches- not Array Formula	Count Matches- not Array Formula
=COMPARE.LISTS($C$6:$C$17,$A$6:$A$17,3)	**=COMPARE.LISTS($C$6:$C$17,$A$6:$A$17,4)**
3	**7**

Using OUTPUT=3 as a single-cell non-array formula gives you a count of the mismatches. This is useful as a simple check that everything in the LookFor list can be found in the LookIn list.

Using OUTPUT=4 as a single-cell non-array formula gives you a count of the matches.

OUTPUT = 3: multi-cell array formula.

	A	B	C	F
5	**LookIn**		**LookFor**	**{=COMPARE.LISTS($C$6:$C$17,$A$6:$A$17,3)}**
6	1		2	**3**
7	2		plug	plug
8			fred	9
9	fred		9	35331
10	FRED		24/09/1996	
11	24/09/1996		23/09/1996	
12	TRUE		FALSE	
13	FALSE			
14	#N/A			
15			#N/A	
16				
17				

This option gives you a count (3 in this case) followed by a list of the **mismatches**.
Cell F9 shows 35331 which is Excel's un-formatted representation of 23/09/1996.

OUTPUT =4: multi-cell array formula

	A	B	C	G
5	**LookIn**		**LookFor**	**{=COMPARE.LISTS($C$6:$C$17,$A$6:$A$17,4)}**
6	1		2	**7**
7	2		plug	2
8			fred	fred
9	fred		9	35332
10	FRED		24/09/1996	FALSE
11	24/09/1996		23/09/1996	
12	TRUE		FALSE	#N/A
13	FALSE			
14	#N/A			
15			#N/A	
16				
17				

This option gives you a count (7) followed by a list of the matches.

Cell G12 shows #N/A because C15 matches A14.

Note that there are 7 matches listed although you can only see 5:
Cell G11 corresponds to the single space at C13 and is not actually empty.
Cell G13 corresponds to the ‘ character at C16 and is not actually empty.

Uniques and Distinct array functions

The 6 functions in the LISTDISTINCTS family provide efficient and flexible methods to work with distinct items or distinct rows in ranges of data.

The LIST functions are either entered as dynamic array formulas, or are nested inside another function that processes the result array. The COUNT functions return a single value.

Options are available for:

- Case-sensitivity (Case_Sense)
- Distinct Rows (ByRows)
- What to show in the unused cells (PadType)
- Sorting the output lists (Sort)
- Ignoring any combination of error values, blanks or zeros (Ignore)

All of the LISTDISTINCTS functions are multithreaded, non-volatile array functions.

The Function Wizard category for the LISTDISTINCTS family is **Statistical**.

LISTDISTINCTS Function

LISTDISTINCTS is an array function that returns an array of the distinct cells or rows in the input data.

Options control what data will be ignored, case sensitivity, sorting the output and padding the output array.

LISTDISTINCTS Syntax

LISTDISTINCTS (theInputData, Ignore, ByRows, Case_Sense, Sort, FillType)

theInputData

theInputData can be a range or an array of constants or an expression returning an array. It identifies the data to be searched for distinct items.

Ignore (Optional)

Controls what cell values will be ignored by the distinct test.
1= Error values ignored. 2 = Blanks or Empty Cells ignored. 4 = Zero values ignored. 8 or larger nothing ignored.
Combinations of Ignore values can be made by adding the values together, although any resulting value greater than 8 means nothing will be ignored.
Default is 3 =1 & 2

ByRows (Optional)

If theInputData is a range or array with multiple columns and rows you may want either to look for distinct rows of data or to look for distinct items in all the cells.
If ByRows is specified as True (True is the default) then each row will be checked against all the other rows for uniqueness. Rows where all the columns contain items to be ignored are not treated as distinct.
If ByRows is False then each cell will be checked against all the other cells for uniqueness. Cells containing items to be ignored are not treated as distinct.
LISTDISTINCTS.SUM and LISTDISTINCT.AVG always work ByRows
The default for ByRows is TRUE.

Case_Sense (Optional)

Specifies whether the comparison will be made in a case-sensitive way (Case_Sense=TRUE) or case will be ignored (Case_Sense=FALSE). The default for Case_Sense is FALSE.

Sort (Optional)

If 0 the output list will be in the same sequence as the theInputData. If 1 the output list of distinct items/rows will be sorted ascending, or if -1 the output list of distinct items/rows will be sorted descending.
For LISTDISTINCTS.COUNT, LISTDISTINCTS.SUM and LISTDISTINCTS.AVG using Sort=2 will sort the column of counts, sums or averages ascending and Sort=-2 will sort them descending.
The default for Sort is 0 (unsorted).

FillType (Optional – default 0 fill with #N/A)

0 (Default) Fill with #N/A
1 Fill with “”
2 Fill with zeros
LISTDISTINCTS.SUM, **LISTDISTINCTS.COUNT** and **LISTDISTINCTS.AVG** are array functions that return **arrays of results**.
The result arrays can either be used as input to other functions or array formulas, entered as dynamic array formulas, or the functions can be entered as multi-cell array formulas so that the result arrays occupy multiple cells.
The functions follow the standard Excel rules for multi-cell array formulas:
If entered into fewer cells than the result array the excess results are not returned.

If entered into more cells than the result array the excess unused cells will be filled with whatever is specified by PadType (0=#N/A, 1="", 2=0, default 0=#N/A)
A single-cell result will be propagated to all the excess unused cells.
A column of results will be propagated to the excess columns
A row of results will be propagated to the excess rows.
COUNTDISTINCTS and **COUNTDUPES** return a single number, so they do not have a PadType option.

Remarks

COUNTDISTINCTS and **COUNTDUPES** These functions return a single number: the total number of distinct items/rows and the total number of duplicated items/rows.
The number of duplicated items/rows is counted as the total count for each non-ignored distinct item/row -1.
They do not need to be entered as array formulas.

LISTDISTINCTS

Unless embedded in another function that will process the array:
If using ByRows the functions should be entered as a dynamic array formula with the same number of columns as the InputData.
If not using ByRows the function should be entered as either a single row or single-column dynamic array formula.

LISTDISTINCTS.COUNT, LISTDISTINCTS.SUM and LISTDISTINCTS.AVG

These functions work the same way as LISTDISTINCTS except that they produce an extra column containing the Counts, Sums and Averages respectively.

LISTDISTINCTS.SUM and LISTDISTINCTS.AVG ignore cells in the SumColumn containing True/False, numbers which are text and text.

LISTDISTINCTS.COUNT Function

LISTDISTINCTS.COUNT outputs a multi-column array of the cells or rows from the input data, where the first column/columns are the distinct items/rows and the last column is the count of that item.
If ByRows is true then the output rows are the distinct rows, but if ByRows is false then the output is 2 columns: the first column is a list of all the distinct items in the input data and the second column is a count for each distinct item.

LISTDISTINCTS Syntax

LISTDISTINCTS.COUNT (theInputData, Ignore, ByRows, Case_Sense, Sort, FillType)

See the LISTDISTINCTS function for an explanation of the parameters.
LISTDISTINCTS.COUNT should be entered as a dynamic array formula.

LISTDISTINCTS.SUM Function

LISTDISTINCTS.SUM outputs a multi-column array of the cells or rows from the input data, where the first column/columns are the distinct items/rows and the last column is the sum of the SumColumn. If ByRows is true then the output rows are the distinct rows, but if ByRows is false then the output is 2 columns: the first column is a list of all the distinct items in the input data and the second column is a sum of the SumColumn for each distinct item.

LISTDISTINCTS.SUM Syntax

LISTDISTINCTS.SUM (theInputData, SumColumn, Ignore, ByRows, Case_Sense, Sort, FillType)

Parameters apart from SumColumn are explained in the LISTDISTINCTS function.

SumColumn

Can be a range or array of constants or an expression returning an array, and must be arranged as a vertical column.
The number of rows should be the same as the number of rows in theInputData.
The values for each corresponding distinct row in theInputData will be summed.
LISTDISTINCTS.SUM and LISTDISTINCTS.AVG ignore cells in the SumColumn containing True/False, numbers which are text and text.

LISTDISTINCTS.AVG Function

LISTDISTINCTS.AVG outputs a multi-column array of the cells or rows from the input data, where the first column/columns are the distinct items/rows and the last column is the count of that item.
If ByRows is true then the output rows are the distinct rows, but if ByRows is false then the output is 2 columns: the first column is a list of all the distinct items in the input data and the second column is a average of the SumColumn for each distinct item.

LISTDISTINCTS.AVG Syntax

LISTDISTINCTS.AVG (theItems, SumColumn, Ignore, ByRows, Case_Sense, Sort, FillType)

Parameters apart from SumColumn are explained in the LISTDISTINCTS function.

SumColumn

Can be a range or array of constants or an expression returning an array, and must be arranged as a vertical column.
The number of rows should be the same as the number of rows in theInputData.
The values for each corresponding distinct row in theInputData will be averaged.
LISTDISTINCTS.SUM and LISTDISTINCTS.AVG ignore cells in the SumColumn containing True/False, numbers which are text and text.

COUNTDISTINCTS Function

The COUNTDISTINCTS function returns a single number: the total number of distinct items/rows.

COUNTDISTINCTS Syntax

COUNTDISTINCTS (theInputData, Ignore, ByRows, Case_Sense)

See the LISTDISTINCTS function for an explanation of the parameters.

COUNTDUPES Function

The COUNTDUPES function returns a single number: the total number of duplicated items/rows.
The number of duplicated items/rows is counted as the total count of items/rows for each non-ignored distinct item/row -1.

COUNTDUPES Syntax

COUNTDUPES (theInputData, Ignore, ByRows, Case_Sense)

See the LISTDISTINCTS function for an explanation of the parameters

LISTDISTINCTS Examples

Sample Data

	A	B	C	D	E
1	**Channel**	**Sector**	**Product**	**Users**	**Spend**
2	Direct	Defense	SpaceCalls	7	1015
3	Direct	Mfg	SpaceCalls	2	205
4	Retail	Defense	SuperPhone	10	100
5	Retail	Mfg	SuperPhone	200	1000
6	direct	Defense	SuperPhone	5	150
7	direct	Mfg	SuperPhone	80	1200

LISTDISTINCTS ByRows

{LISTDISTINCTS($A$2:$B$7)}		
direct	Defense	
direct	Mfg	
Retail	Defense	
Retail	Mfg	
#N/A	#N/A	
#N/A	#N/A	

Each distinct row in the input is shown. The extra 2 rows are padded with #N/A.

LISTDISTINCTS.COUNT ByRows, filled with blanks

LISTDISTINCTS.COUNT($A$2:$B$7,,,,,1)			
direct	Defense	2	
direct	Mfg	2	
Retail	Defense	1	
Retail	Mfg	1	

Each distinct row in the input is shown with a count of the number of occurrences. The extra 2 rows are padded with blanks.

LISTDISTINCTS.COUNT ByRows=False, filled with blanks

LISTDISTINCTS.COUNT($A$2:$B$7,,FALSE,,,1)				
Direct	4			
Defense	3			
Mfg	3			
Retail	2			

Each distinct item in the Input Data is listed with a count of the number of ocurrences, the extra 2 rows and column are filled with blanks.

LISTDISTINCTS.SUM filled with blanks

LISTDISTINCTS.SUM($A$2:$B$7,$E$2:$E$7,,,,1)		
direct	Defense	1165
direct	Mfg	1405
Retail	Defense	100
Retail	Mfg	1000

The distinct rows are listed with the sum of column E (Spend)

LISTDISTINCTS.AVG filled with blanks

LISTDISTINCTS.AVG($A$2:$B$7,$E$2:$E$7,,,,1)		
direct	Defense	582.5
direct	Mfg	702.5
Retail	Defense	100
Retail	Mfg	1000

The distinct rows are listed with the average of their Column E Spend.

COUNTDISTINCT ByRows. Single cell formula

COUNTDISTINCTS($A$2:$B$7)
4

There are 4 distinct rows in A2:B7

COUNTDUPES ByRows. Single cell formula

COUNTDUPES($A$2:$B$7)
2

There are only 2 duplicates in A2:B7 (2 Rows appear twice)

COUNTDISTINCT ByRows=False

COUNTDISTINCTS($A$2:$C$7,,FALSE)
6

There are 6 distinct items in A2:C7

LISTDISTINCT ByRows=False

LISTDISTINCTS(A2:C7,,,FALSE)
Direct
Defense
SpaceCalls
Mfg
Retail
SuperPhone

The 6 distinct items in A2:C7 are listed

COUNTDUPES ByRows=False

COUNTDUPES($A$2:$C$7,,FALSE)
12

There are 12 duplicated items in A2:C7
(3 x Direct, 1 x Retail, 2 x Defense, 2 x Mfg, 1x SpaceCalls, 3 x SuperPhone)

Array Sorting Functions

FastExcel SpeedTools has 6 dynamic sorting functions that can be used to return sorted arrays.

- VSORTC - Sort of a vertical range/array, not case sensitive, using Excel's sorting rules
- VSORTC.INDEX - Index sort of a vertical range/array, not case sensitive, using Excel's sorting rules
- Case.VSORTC - Sort of a vertical range/array, case sensitive, lower-case before upper-case, using Excel's sorting rules
- Case.VSORTC.INDEX – Index sort of a vertical range/array, case sensitive, lower-case before upper-case, using Excel's sorting rules
- VSORTB - Fast sort of a vertical range, case sensitive, upper-case before lower-case, ignores collating rules for accented characters
- VSORTB.INDEX - Fast Index sort of a vertical range, case sensitive, upper-case before lower-case, ignores collating rules for accented characters

The sorting functions use a stable sort method which preserves the original order of equivalent items.

Up to 15 columns in the input data can be sorted using a mixture of ascending and descending sort sequence.

Collating Sequences

A collating sequence defines the order in which characters and values are sorted. Collating sequences tend to vary by country, and sometimes a given country will have more than one collating sequence available.
There are 2 different types of text collating sequences available in the FastExcel SpeedTools sorting functions.
The **locale-dependent collating sequence** uses the user-specified National Language locale sequence. This handles national characters and character combinations based on the conventions established for each locale. This is the method used by Excel SORT and all the LOOKUP functions.
The VSORTC, VSORTC.INDEX, Case.VSORTC, Case.VSORTC.INDEX functions all use the locale-dependent collating sequence, and follow Excel's sorting rules for hyphens and apostrophes.
The **collating sequence** used by VSORTB and VSORTB.INDEX is faster for sorting text than the locale-dependent sequence but may not give results compatible with Excel's SORT and LOOKUP functions using the sorted option. VSORTB and VSORTB.INDEX ignore Excel's sorting rules for hyphens and apostrophes.
Excel's collating sequence by data type is:
Numbers < Textual Numbers < Text < Logical < Error Values < Empty cells (always last)
Empty cells are returned as zero by the SpeedTools sorting functions.

Sort and Index Sort

A conventional sort returns the input data in sorted order.
An Index Sort returns the position or index in the input data of the nth item in the sort order.

- Sort(B, C, A, D) gives A,B,C,D
- Index Sort (B, C, A, D) gives 3,1,2,4.
  First A is position 3, second B is position 1, third C is position 2, fourth D is position 4.

Case-Sensitive Sort

The locale-dependent SORTs have both case-sensitive and case-insensitive versions. Binary Sorts are always case-sensitive:

I	J	K	L	M	N
Data	VSORT	Case.VSORT	VSORTB	Excel	Excel Case-sensitive
B	a	a	A	a	a
b	A	A	B	A	A
a	B	b	a	B	b
A	b	B	b	b	B

- VSORTC is not case-sensitive so the relative positions of the equivalent upper and lower case letters is preserved from the original sequence.
- Case.VSORTC is case-sensitive and sorts lower-case before upper-case in the same way as Excel.
- VSORTB is also case-sensitive but sorts upper-case before lower-case (this sequence is dependent on the code-page and character set being used).

VSORTC – Dynamic text collating Sort of a vertical range or array

This function sorts a vertical array or a range containing one or more columns. The sort is NOT case-sensitive and is done using a locale-based text collating sequence which respects national language characters. The output sequence will be the same as a non-case-sensitive EXCEL SORT.
VSORTC is a non-volatile multi-threaded multi-cell array function.

VSORTC Syntax

VSORTC(theInputData, SortColumn1, SortColumn2, … SortColumn15))

The first parameter is required, all other parameters are optional.
The output from VSORTC will be a vertical sorted array. The number of rows sorted and output will be the smaller of the number of rows in the input data and the number of rows in the used range. The number of columns will be the number of columns in the input data.

TheInputData (required)

The data to be sorted, given as a vertical array of constants or a calculated range or a range. The range or array can contain as many columns as required.

SortColumn1 … SortColumn15 (optional)

Gives the index column number(s) of the columns to be used as sort keys. Up to 15 sort keys can be specified.
Positive column numbers will be sorted ascending and negative column numbers will be sorted descending. You can have both positive and negative numbers in the same function call. 1 denotes the first column in the input data. If all Sort Column parameters are omitted all columns in the input data will be used as ascending sort keys.

VSORTC Examples

VSORTC(A:C) will sort column A to C ascending, using the minimum of the number of rows in the used range and the number of rows in the multi-cell array formula that contains VSORTC. If the formula containing VSORTC is only entered into a single cell then VSORTC will return the number of rows in the used range.

- VSORTC ({4,3,1,2}) returns a single row with 4 columns containing 4,3,1,2 (the input data is a single row).
- VSORTC({4;3;1;2}) returns a column with 4 rows containing 1,2,3,4
- VSORTC(A1:C100000,1,-2) will return 100000 rows by 3 columns sorted ascending on column A and descending on column B.

I	J	K	L	M	N
Data	VSORT	Case.VSORT	VSORTB	Excel	Excel Case-sensitive
B	a	a	A	a	a
b	A	A	B	A	A
a	B	b	a	B	b
A	b	B	b	b	B

Case.VSORTC – Case-sensitive dynamic Sort of a vertical range or array

This function sorts a vertical array or a range containing one or more columns. The sort IS case-sensitive and is done using a locale-based collating sequence which respects national language characters. The output sequence will be the same as a case-sensitive EXCEL SORT.
VSORTC is a non-volatile multi-threaded multi-cell array function.

Case.VSORTC Syntax

Case.VSORTC(theInputData, SortColumn1, SortColumn2, … SortColumn15))

The first parameter is required, all other parameters are optional.
The output from Case.VSORTC will be a vertical sorted array. The number of rows sorted and output will be the smaller of the number of rows in the input data and the number of rows in the used range. The number of columns will be the number of columns in the input data.
TheInputData (required)
The data to be sorted, given as a vertical array of constants or a calculated range or a range. The range or array can contain as many columns as required.

SortColumn1 … SortColumn15 (optional)

Gives the index column number(s) of the columns to be used as sort keys. Up to 15 sort keys can be specified.
Positive column numbers will be sorted ascending and negative column numbers will be sorted descending. You can have both positive and negative numbers in the same function call. 1 denotes the first column in the input data.
If all Sort Column parameters are omitted all columns in the input data will be used as ascending sort keys.

Case.VSORTC Examples

- Case.VSORTC(A:C) will sort column A to C ascending, using the minimum of the number of rows in the used range and the number of rows in the multi-cell array formula that contains Case.VSORTC. If the formula containing Case.VSORTC is only entered into a single cell then Case.VSORTC will return the number of rows in the used range.
- Case.VSORTC({B;b;a;A}) returns a column with 4 rows containing a, A, b, B (lower case before upper case).
- Case.VSORTC(A1:C100000,1,-2) will return 100000 rows by 3 columns sorted ascending on column A and descending on column B.

I	J	K	L	M	N
Data	VSORT	Case.VSORT	VSORTB	Excel	Excel Case-sensitive
B	a	a	A	a	a
b	A	A	B	A	A
a	B	b	a	B	b
A	b	B	b	b	B

VSORTB – Fast Dynamic Sort of a vertical range or array

This function sorts a vertical array or a range containing one or more columns. The sort is case-sensitive and is done using a binary collating sequence which is dependent on the positions of the characters in the code-page. **The output sequence may NOT be the same as EXCEL SORT for text items.**
VSORTB is a non-volatile multi-threaded multi-cell array function.

VSORTB Syntax

VSORTB(theInputData, SortColumn1, SortColumn2, … SortColumn15))

The first parameter is required, all other parameters are optional.
The output from VSORTB will be a vertical sorted array. The number of rows sorted and output will be the smaller of the number of rows in the input data and the number of rows in the used range. The number of columns will be the number of columns in the input data.
TheInputData (required)

The data to be sorted, given as a vertical array of constants or a calculated range or a range. The range or array can contain as many columns as required.

SortColumn1 … SortColumn15 (optional)

Gives the index column number(s) of the columns to be used as sort keys. Up to 15 sort keys can be specified.
Positive column numbers will be sorted ascending and negative column numbers will be sorted descending. You can have both positive and negative numbers in the same function call.
1 denotes the first column in the input data.
If all Sort Column parameters are omitted all columns in the input data will be used as ascending sort keys.

VSORTB Examples

VSORTB(A:C) will sort column A to C ascending, using the minimum of the number of rows in the used range and the number of rows in the multi-cell array formula that contains VSORTB. If the formula containing VSORTB is only entered into a single cell then VSORTB will return the number of rows in the used range.

- VSORTB ({4,3,1,2}) returns a single row with 4 columns containing 4,3,1,2 (the input data is a single row).
- VSORTB({4;3;1;2}) returns a column with 4 rows containing 1,2,3,4
- VSORTB(A1:C100000,1,-2) will return 100000 rows by 3 columns sorted ascending on column A and descending on column B.

I	J	K	L	M	N
Data	VSORT	Case.VSORT	VSORTB	Excel	Excel Case-sensitive
B	a	a	A	a	a
b	A	A	B	A	A
a	B	b	a	B	b
A	b	B	b	b	B

VSORTC.INDEX – Collating Text Index Sort of a vertical range or array

This function index sorts a vertical array or a range containing one or more columns. The sort is NOT case-sensitive and is done using a locale-based collating sequence which respects national language characters and Excel's sorting rules. The output sequence will be the same as a non-case-sensitive EXCEL SORT.

VSORTC.INDEX is a non-volatile multi-threaded multi-cell array function.

VSORTC.INDEX Syntax

VSORTC.INDEX(theInputData, SortColumn1, SortColumn2, … SortColumn15))

The first parameter is required, all other parameters are optional.

The output from VSORTC.INDEX will be a vertical sorted array of index numbers. The index numbers give the relative position in the input data of the nth item in the sorted output, rather than the sorted output itself.

The number of rows sorted and output will be the smaller of the number of rows in the input data and the number of rows in the used range.

The number of columns will always be 1.

TheInputData (required)

The data to be sorted, given as a vertical array of constants or a calculated range or a range. The range or array can contain as many columns as required.

SortColumn1 … SortColumn15 (optional)

Gives the index column number(s) of the columns to be used as sort keys. Up to 15 sort keys can be specified.

Positive column numbers will be sorted ascending and negative column numbers will be sorted descending. You can have both positive and negative numbers in the same function call.

1 denotes the first column in the input data.

If all Sort Column parameters are omitted all columns in the input data will be used as ascending sort keys.

VSORTC.INDEX Examples

- VSORTC.INDEX({B;b;a;A}) returns a column with 4 rows containing 3, 4, 1, 2 (first is the 3rd in the input data ("a") then the 4th ("A"), then the 1st ("B"), then the 2nd ("b")) .
- VSORTC.INDEX(A1:C100000,1,-2) will return 100000 rows by 1 column of index numbers sorted ascending on column A and descending on column B.

Data	VSORT.INDEX	Case.VSORT.INDEX	VSORTB.INDEX
B	3	3	4
b	4	4	1
a	1	2	3
A	2	1	2

Case.VSORTC.INDEX – Collating Text Index Sort of a vertical range or array

This function index sorts a vertical array or a range containing one or more columns. The sort IS case-sensitive and is done using a locale-based collating sequence which respects national language characters and Excel's sorting rules. The output sequence will be the same as a case-sensitive EXCEL SORT.

Case.VSORTC.INDEX is a non-volatile multi-threaded multi-cell array function.

Case.VSORTC.INDEX Syntax

Case.VSORTC.INDEX(theInputData, SortColumn1, SortColumn2, … SortColumn15))

The first parameter is required, all other parameters are optional.

The output from Case.VSORTC.INDEX will be a vertical sorted array of index numbers. The index numbers give the relative position in the input data of the nth item in the sorted output, rather than the sorted output itself.

The number of rows sorted and output will be the smaller of the number of rows in the input data and the number of rows in the used range.

The number of columns will always be 1.

TheInputData (required)

The data to be sorted, given as a vertical array of constants or a calculated range or a range. The range or array can contain as many columns as required.

SortColumn1 … SortColumn15 (optional)

Gives the index column number(s) of the columns to be used as sort keys. Up to 15 sort keys can be specified.

Positive column numbers will be sorted ascending and negative column numbers will be sorted descending. You can have both positive and negative numbers in the same function call.

1 denotes the first column in the input data.

If all Sort Column parameters are omitted all columns in the input data will be used as ascending sort keys.

Case.VSORTC.INDEX Examples

- Case.VSORTC.INDEX({B;b;a;A}) returns a column with 4 rows containing 3, 4, 2, 1 (first is the 3rd in the input data ("a"), then the 4th ("A"), then the 2nd ("b" then the 1st ("A")) (case-sensitive collate means lower case before upper case so the a comes before A).
- Case.VSORTC.INDEX(A1:C100000,1,-2) will return 100000 rows by 1 column of index numbers sorted ascending on column A and descending on column B.

Data	VSORT.INDEX	Case.VSORT.INDEX	VSORTB.INDEX
B	3	3	4
b	4	4	1
a	1	2	3
A	2	1	2

VSORTB.INDEX – Fast Index Sort of a vertical range or array

This function index sorts a vertical array or a range containing one or more columns. The sort is case-sensitive and is done using a binary collating sequence which is dependent on the positions of the characters in the code-page ignoring Excel's rules for sorting apostrophes and hyphens. **The output sequence may NOT be the same as EXCEL SORT for text items.**
VSORTB is a non-volatile multi-threaded array function.

VSORTB.INDEX Syntax

VSORTB.INDEX(theInputData, SortColumn1, SortColumn2, … SortColumn15))

The first parameter is required, all other parameters are optional.
The output from VSORTB.INDEX will be a vertical sorted array of index numbers. The index numbers give the relative position in the input data of the nth item in the sorted output, rather than the sorted output itself.
The number of rows sorted and output will be the smaller of the number of rows in the input data and the number of rows in the used range.
The number of columns will always be 1.

TheInputData (required)

The data to be sorted, given as a vertical array of constants or a calculated range or a range. The range or array can contain as many columns as required.

SortColumn1 … SortColumn15 (optional)

Gives the index column number(s) of the columns to be used as sort keys. Up to 15 sort keys can be specified.
Positive column numbers will be sorted ascending and negative column numbers will be sorted descending. You can have both positive and negative numbers in the same function call.
1 denotes the first column in the input data.
If all Sort Column parameters are omitted all columns in the input data will be used as ascending sort keys.

VSORTB.INDEX Examples

- VSORTB.INDEX({B;b;a;A}) returns a column with 4 rows containing 3, 4, 2, 1 (first is the 3rd in the input data ("a"), then the 4th ("A"), then the 2nd ("b" then the 1st ("A")) (case-sensitive collate means lower case before upper case so the a comes before A).
- VSORTB.INDEX(A1:C100000,1,-2) will return 100000 rows by 1 column of index numbers sorted ascending on column A and descending on column B.

Data	VSORT.INDEX	Case.VSORT.INDEX	VSORTB.INDEX
B	3	3	4
b	4	4	1
a	1	2	3
A	2	1	2

Logical Functions for Array Formulas

Excel provides the AND and OR functions to allow you to specify multiple conditions in a Formula or IF statement.
But Excel's AND/OR functions evaluate all of the given arguments/parameters and return a single TRUE/FALSE answer. This makes them difficult to use in array formulas where you usually require a column, row or array of True/False answers.
SpeedTools provides a family of array-formula friendly AND/OR functions:

OR.ROWS, OR.COLS, OR.CELLS, AND.ROWS, AND.COLS, AND.CELLS, ALL, ANY, NONE

The ROWS functions evaluate each row in the input arguments separately to provide a column of True/False answers. This is the most frequently used flavour for array formulas.
The COLS functions evaluate each column in the input arguments separately to provide a row of True/False answers.
The CELL function evaluates each corresponding element in the input arguments separately to provide a 2-dimensional array of True/False answers.
The functions can be nested together to provide complex logical array expressions.
The functions are non-volatile, multi-threaded array functions.

Differences to Excel's AND OR functions

SpeedTools Logical Functions	**Excel Logical Functions**
No True/False results found=False	No True/False results found=#Value
Numeric Text <>0 = True	Numeric Text <>0 = False
Returns rows, columns or arrays	Returns a single True/False

ALL, ANY, NONE SpeedTools Logical Functions for Ordinary Formulas

To provide compatibility with the SpeedTools array functions SpeedTools also provides three logical functions for use in non-array formulas:
ALL which is equivalent to AND
ANY which is equivalent to OR
NONE which is equivalent to NOT OR
These 3 functions handle Numeric Text and the absence of any True/False results in the same way as the SpeedTools logical array functions.

AND.ROWS, AND.COLS, AND.CELLS, OR.ROWS, OR.COLS, OR.CELLS Syntax

OR.ROWS(Logicaltest1, logicaltest2, …)

OR.COLS(Logicaltest1, logicaltest2, …)

OR.CELLS(Logicaltest1, logicaltest2, …)

AND.ROWS(Logicaltest1, logicaltest2, …)

AND.COLS(Logicaltest1, logicaltest2, …)

AND.CELLS(Logicaltest1, logicaltest2, …)

LogicalTest

- Each LogicalTest parameter for these functions can be a constant, a constant array, a range or a formula.
- For OR.ROWS and AND.ROWS the number of rows in each LogicalTest parameter must be the same, but the number of columns may be different.
- For OR.COLS and AND.COLS the number of columns in each LogicalTest parameter must be the same, but the number of rows may be different.
- For OR.CELLS and EACH.CELL the number of rows and columns must be the same in all LogicalTest parameters.
- If there are any error values in any of the parameters the first error value found is returned instead of True or False.
- False, alphabetic text, empty, 0 and 0 as text are treated as False
- True, non-zero numbers and numeric text are treated as True (Excel's OR function treats numeric text as False)
- If no True/False or expressions resolving to True/False are found the result is False (Excel's OR function returns #Value)

Examples of SpeedTools Logical Functions:

Row Functions

The OR.ROWS formula is entered in the column as

=OR.ROWS(A2:A21,B2:B21,C2:C21)

The AND.ROWS formula is entered in the column as

=AND.ROWS(A2:A21,B2:B21,C2:C21)

The functions have evaluated each row of the data to produce a corresponding True/False

Data			***OR.ROWS***	***AND.ROWS***
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE
fred	joe	bill	FALSE	FALSE
1	FALSE	FALSE	TRUE	FALSE
FALSE	-1	FALSE	TRUE	FALSE
FALSE	FALSE	0.001	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	joe	bill	TRUE	FALSE
fred	TRUE	bill	TRUE	FALSE
fred	joe	TRUE	TRUE	FALSE
FALSE	joe	bill	FALSE	FALSE
fred	FALSE	bill	FALSE	FALSE
fred	joe	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	FALSE
TRUE	0	FALSE	TRUE	FALSE
			FALSE	FALSE
			FALSE	FALSE

Column Functions:

Data		
TRUE	FALSE	FALSE
1	TRUE	FALSE
-1	FALSE	0
1	TRUE	0

=AND.COLS(A55:C58}		
TRUE	FALSE	FALSE
=OR.COLS(A55:C58)		
TRUE	TRUE	FALSE

The functions have evaluated each column of the data to produce a corresponding True/False

CELL Functions:

=OR.CELLS($A$2:$C$21,$E$2:$G$19)

=AND.CELLS($A$2:$C$21,$E$2:$G$19)

The functions have evaluated each corresponding element in the 2 data tables to produce a corresponding array of True/False.

Data			More Data			OR.CELLS			AND.CELLS		
TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
fred	joe	bill	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE
1	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	-1	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE
FALSE	FALSE	0.001	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE
TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE
TRUE	joe	bill	fred	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
fred	TRUE	bill	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE
fred	joe	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE
FALSE	joe	bill	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
fred	FALSE	bill	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
fred	joe	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE
TRUE	0	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
						FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
						FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Comparison of ANY ALL NONE with Excel AND/OR

Differences are highlighted in yellow.

Data		*OR*	*AND*	*OR.ROWS*	*ANY*	*ALL*	*NONE*
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
99	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE
-99	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE
0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
24/09/1943	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE
1	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
#DIV/0!	FALSE	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
#N/A	FALSE	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
#NAME?	FALSE	#NAME?	#NAME?	#NAME?	#NAME?	#NAME?	#NAME?
#NULL!	FALSE	#NULL!	#NULL!	#NULL!	#NULL!	#NULL!	#NULL!
#NUM!	FALSE	#NUM!	#NUM!	#NUM!	#NUM!	#NUM!	#NUM!
#REF!	FALSE	#REF!	#REF!	#REF!	#REF!	#REF!	#REF!
#VALUE!	FALSE	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!
	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
fred	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
fred	joe	#VALUE!	#VALUE!	FALSE	FALSE	FALSE	TRUE
22	11	#VALUE!	#VALUE!	TRUE	TRUE	TRUE	FALSE

Array and Range Filtering Functions

The filtering functions provide extended ways of using one or more criteria to filter out subsets of data.
Very fast performance is achieved for sorted data.
The functions can be used as dynamic array formulas to return the data subsets directly.
They can also be used within any aggregating function (SUM, MEDIAN, RANK etc to provide the equivalent MEDIANIFS, RANKIFS function.

- FILTERX - Extended progressive version of FILTER with *IFS criteria
- ACOUNTIFS - count using extended multiple conditions
- ASUMIFS - sum using extended multiple conditions
- FILTER.IFS - filter out subsets of data using multiple extended conditions
- FILTER.SORTED - filter out subsets of sorted data using multiple extended conditions
- FILTER.MATCH - filter out row numbers of the data using multiple extended conditions
- FILTER.VISIBLE - filter out the visible rows
- Rgx.SUMIF - sum values using Regular Expressions
- Rgx.COUNTIF - count values using Regular Expressions
- INSPAN - Returns an array of True/False for a span within a range or array
- SPAN - Extracts a span of rows and columns from a range or array
- MATCH.SPAN - Returns the first and last row or column numbers for a span within a range or array.

All these functions except FILTER.VISIBLE handle both ranges and arrays as inputs.

The FILTER.IFS Multiple Criteria Function Family

The FILTER.IFS, FILTER.SORTED, FILTER.MATCH, ASUMIFS and ACOUNTIFS functions are a family of high-performance SpeedTools functions you can use to replace many SUMPRODUCT, FILTER and array functions.

- The FILTER.IFS functions are extremely fast when used on sorted data or well-structured data.
- Data can be sorted on ascending or descending criteria, or unsorted.
- Use FILTER.IFS inside functions like RANK, MAX, MIN, SUM, COUNT, COUNTA, AVERAGE, MEDIAN, MODE, LARGE, INDEX (any function or UDF that will handle a range or array input), or in a multi-cell array formula to return the resulting filtered subset of data.
- Condition/Criteria operators can be
  - Relational Operators =, >=, <=, >, <, <>, True, False
  - ~ means Like pattern matching (VBA Like), ~~ means regular expression matching
  - All Criteria can be inverted using the Not prefix ¬
  - A list of alternative inclusion or exclusion conditions can be given as a calculation, array or Range Reference
  - Data Type Criteria filters #ERR, #TXT, #N, #BOOL, #EMPTY, #ZLS, #TYPE, #BLANK
- The Criteria column being compared to the Condition/criteria can be
  - A column range within or outside the RangeArray
  - A Calculated column (an expression that evaluates to a column array)
  - An array
- A row is included when ALL of a set of conditions are met.
- Sets of Conditions can be separated into #OR# blocks
- Columns within the RangeArray can be identified either by a label in the first row, or by a column number, or directly using a range reference or array expression.
- Up to 18 pairs of Criteria Columns and their Criteria may be used.

Optimizing the performance of the FILTER.IFS family of functions

FILTER.IFS is faster with sorted criteria data than unsorted.

FILTER.IFS will exploit grouping or structuring of unsorted data.

The sequence in which the Criteria/Conditions are given can have a significant influence on performance.

- Where some criteria apply to the sorted column(s) and some are unsorted, specify the sorted ones first.
- For both Sorted and Unsorted data the first criteria should be the one that eliminates the largest number of rows.
- For both sorted and unsorted criteria specify = criteria before other types of criteria.
- Where there are 2 criteria applying to the same column (for example data between 2 dates) the criteria should be adjacent.
- For Unsorted data FILTER.IFS will be most efficient when the first criterion results in a relatively small number of groups of rows meeting the first criteria
- The worst case for FILTER.IFS is unsorted data where the first criterion eliminates every other row.

FILTER.IFS function

FILTER.IFS returns the cells from the chosen column(s) in the input data that match the extended conditions or criteria. The syntax for the criteria is similar to SUMIFS and COUNTIFS but with considerably more options.
FILTER.IFS can be used either to add multiple conditions to aggregating functions like LISTDISTINCTS.SUM, SUM, MEDIAN, RANK etc. or as a multi-cell array formula.
FILTER.IFS is most efficient when used with data that is sorted on the first few criteria columns.

FILTER.IFS Syntax

FILTER.IFS(nSortedCrits,RangeArray,ReturnCol,CriteriaColumn1,Criteria1, CriteriaColumn2,Criteria2, ..., ["#OR#", nSortedCrits,] CriteriaColumnx, Criteriax, ...)

Example: SUM(FILTER.IFS(3,$A$3:$F$10303,"Value","FromCtry","=" & $H6,"ToCtry","=" & $I6,"ToCity","=" & $J6,"Date",">"&$K6,"Date","<"&$L6))

FILTER.IFS Parameters

nSortedCrits (required for all the FILTER.IFS family functions except FILTER.SORTED, FILTER.VISIBLE and FILTERX)

If nSortedCrits is NOT equal to zero the RangeArray must be sorted on the criteria columns in the DataRange.

Gives the number of sorted Criteria columns in the RangeArray.
If nSortedCrits is 2 then CriteriaColumn1 must be the major sort column in the RangeArray, CriteriaColumn2 the next minor sort column in the RangeArray, and so on. (In other words RangeArray must be sorted by CriteriaColumn2 within CriteriaColumn1).
The sorted criteria columns must all be sorted either Ascending or Descending. Positive nSortedCrits indicates sorted ascending, negative nSortedCrits indicates descending and zero indicates that the criteria columns are not sorted.
Columns outside the RangeArray and Calculated Columns are always treated as unsorted.
Columns with criteria operators ~, ~~, the Type criteria operators or with a list of criteria given as an array constant or multi-cell range reference are always treated as unsorted.
The first column with a criteria operator other than =, True, False will cause subsequent columns to be treated as unsorted.
Using non-zero NSortedCrits with unsorted criteria data will give unpredictable and usually incorrect results.

RangeArray (required)

The RangeArray can refer to a Range or an array or the result of an expression.

The RangeArray may optionally contain column labels in the first row, in which case at least one of the criteria columns or return column should be identified using a label. If you only use column numbers then the RangeArray should NOT include labels.

ReturnCol (required for all the FILTER.IFS family functions except FILTER.MATCH and FILTERX)

Specifies the column in RangeArray from which the subset of data will be returned. If given as text will be interpreted as a column label in the first row of the RangeArray.

If given as a number will be interpreted as a column number within RangeArray, and RangeArray should NOT include a header row.

If ReturnCol is 0 then all the columns in RangeArray will be returned (ASUMIFS and ACOUNTIFS return a row of sums and counts for each column).

CriteriaColumn1 (required)

Specifies the first Criteria Column. Can be text, a number, a Range Reference, an array of constants or an expression that results in a column array.
Text will be interpreted as a column label in the first row of the RangeArray
Number will be interpreted as a column number within RangeArray.
Range Reference or Array will be interpreted as an independent column outside RangeArray.
An array of constants will be interpreted as a column of data.
An expression will be evaluated by Excel as a calculated column before being passed to the FILTER function.
Criteria Columns must contain the same number of rows as the RangeArray, but allowance is made for a row of column labels in either or both the RangeArray and Criteria Columns.

Criteria1 (required)

Specifies the criteria expression to be applied to CriteraColumn1.
The Criteria expression must resolve to a text string starting with an optional criteria operator and ending with a Criteria value.
If no criteria operator is found at the start then = will be assumed.
A Criteria expression may be any Excel expression containing strings, range references, defined names, functions and operators

The maximum number of pairs of Criteria Columns and their criteria is 18.

Criteria Operators

Valid Criteria operators are:

The relational operators:

- = is equal to
- > is greater than
- >= is greater than or equal to
- < is less than
- <= is less than or equal to
- ¬ is not equal to, also negates following criteria operators
- If no operator is given it is treated as =, except that the data type is never converted (see Data Types section below).

The Pattern operators:

- ~ Like the criteria value pattern using the wild-card characters * and ?
- ~~ Matches against a regular expression pattern

The Boolean operators:

- True the cell value is true
- False the cell value is false

Because Excel propagates error values through formulas and expressions all the above criteria operators have no effect when preceding an error value.

Criteria Lists

Criteria can be given either as a single criterion or a list of alternative Criteria.

The list of alternative Criteria can be either an expression returning an array, an array of Constants or a Range Reference.

The list of alternative criteria can either be:

- An **Inclusion list** – a row that matches any of the items in the list will be included.
- An **exclusion list** – a row that matches any of the items in the list will be excluded. The first element in the list must be two not symbols ¬¬ entered as a string.
- An **All list** – row will must match All of the items in the list to be included. The first element in the list must be && (two & characters) entered as a string. This can be used for example to match between 2 dates.

The individual elements in the array or cells in the Range Reference may each start with their individual criteria Operator, or may not have a criteria operator.

Criteria Lists Examples:

FILTER.IFS(0,$A$1:$C$6,"Hundreds","Digits",{"Four","One","Two"},"Tens",{"Fifty","Fourty","Thirty","Twenty"})

The Digits column must contain any of Four, One or Two and the Tens column must contain any of Fifty, Forty, Thirty or Twenty.

FILTER.IFS(0,$A$1:$C$6,"Hundreds","Digits",{"Four","One","Two"},"Tens",{“¬¬”,"Fifty","Fourty","Thirty","Twenty"})

The Digits column must contain any of Four, One or Two and the Tens column must NOT contain any of Fifty, Forty, Thirty or Twenty.

FILTERX(tblSalesData,tblSalesData[ShipDate],VSTACK(“&&”, ”>=” & Date(2026,9,24), “<=” & Date(2026,12,31)))

The Sales Data table will be filtered for sales with ship dates between 24 Sep 2026 and 31 Dec 2026 inclusive.

For very large lists it will be more efficient to use COMPARE.LISTS:

FILTERX(tblSalesData,tblSalesData[ShipDate],VSTACK(“&&”, ”>=” & Date(2026,9,24), “<=” & Date(2026,12,31)),COMPARE.LISTS(tblSalesData[Products),HighValueProducts],True)

The Sales Data table will be filtered for High Value Products sales with ship dates between 24 Sep 2026 and 31 Dec 2026 inclusive.

The Data Type Operators:

- #TYPE the data type of the cell is the same as the first data cell in the criteria column
- #ERR the cell contains an error value
- #TXT the cell contains a string (text) value
- #N the cell contains a number (integer or with decimal point, date, time, currency)
- #BOOL the cell contains a Boolean True or False
- #EMPTY the cell is empty
- #ZLS the cell contains a zero length string
- #BLANK the cell contains a string of 1 or more blanks/spaces

Any of these criteria operators can be prefixed by ¬ (not) to convert the operator from an inclusion operator to an exclusion operator.

Data Type Comparison

Excel has 5 fundamental data types: Number (integers, real numbers, dates, times, currency), String (text), Boolean (true or false), Error (#NA etc) and Empty (unused cells).
When a column of mixed data types is sorted Excel uses the following comparison relationships:
Numbers < Strings < Booleans < Errors
Empty cells are always sorted last, both in an ascending and a descending sort.
The SpeedTools FILTER functions use the same comparison relationship between data types.

Data Type Conversion

When a criteria value is preceded by any of the criteria operators its data type is not known (Criteria values without any preceding criteria operator have a specific data type).
For example, in "<1234" the 1234 could be a number or could be a string. When this criteria value is being compared to a criteria column that could contain multiple data types the FILTER.IFS functions use the following rules:
When the criteria operator is ~ or ~~ the criteria value and the values in the criteria column will be converted to strings before doing the pattern match.
Otherwise **the preferred data type will be the data type of the first data (non-header) cell in the criteria column**. FILTER.IFS will attempt to convert the criteria value to the preferred data type. If this is not possible FILTER.IFS will convert the criteria value to a number, then a string, then a Boolean then an error.

Overriding Data Type Conversion using the & Prefix

If your criteria column contains mixed data types (for instance both numeric and string numbers) you can ask FILTER.IFS to try to convert the criteria value to the best possible data type match with each cell in the criteria column.
So FILTER.IFS would use the numeric version of the criteria value to compare with a numeric number and the string version of the criteria value to compare with a string/text number.
You request this by adding a & character as a prefix to the criteria operator (after the ¬ prefix if you are using the exclusion prefix).

Criteria Values

Criteria values and Criteria column values are NOT Case sensitive.

Criteria Like Patterns

Like Patterns can contain:

- * Any number of characters, including none
- ? Any single character

Pattern Examples:

Column Value	Criteria	Result	Criteria Explanation
"aBBBa"	"~a*a"	True	String starting with a followed by any characters and ending with a
"a2a"	"~a?a"	True	String starting with a followed by any single character followed by a
"BAT123khg"	"~B?T*"	True	B followed by any single character followed by T followed by any or no characters
"CAT123khg"	"~B?T*"	False	B followed by any single character followed by T followed by any or no characters

CriteriaColumn2, Criteria2, … (optional)

You can give additional pairs of Criteria Column and Criteria.

All of the Criteria pairs given in a set must be True for a row of data from the Return Column to be included.

"#OR#", nSortedCriteria, (optional)

If you need alternative sets of criteria you can separate them with "#OR#", nSortedCrits.

A row will be selected from the Return Column if ANY of the alternative sets of Criteria are met.

The row will only be selected once if both alternative sets of criteria are met (no double-counting).

For Example:

CriteriaCol1,Criteria1,CriteriaCol2, Criteria2, "#OR#",1,CriteriaCol3, Criteria3,CriteriaCol4,Criteria4

A row will be selected from the Return Column if

(both criteria1 and Criteria2 are True) OR (both Criteria3 and Criteria4 are True)

Note that a Criterion can also contain a list of alternative conditions (see above)

Combining FILTER.IFS Criteria in Logical Combinations

Default is AND

By default, ALL the criteria given must be met for a row to be included in the subset.

Multiple alternatives for a single column

You can use criteria lists to give a number of alternative or combined criteria for a single column (Inclusive, Exclusive or All lists).

Alternative Sets of Criteria

Where you have alternative sets of criteria (both Criteria1 and Criteria2 must be True) OR (both Criteria3 and Criteria4 must be True) you can use **#OR#** to separate them.

More complex Logical Combinations

You can use SpeedTools's OR.ROWS and AND.ROWS functions to build more complicated calculated combinations of criteria within FILTER (see below).

FILTER.IFS and ASUMIFS Examples

These examples show some of ways that FILTER.IFS and ASUMIFS can be used to calculate dynamic statistics from a data set.

	A	B	C	D	E	F
1	**Test Data, sorted FromCtry,ToCtry, toCity: 10300 rows**					
2						
3	**FromCtry**	**ToCtry**	**FromCity**	**ToCity**	**Date**	**Value**
4	AT	BE	graz	brussels	26 February 2000	152
5	AT	BE	oth AT	brussels	15 June 2000	476
6	AT	BE	salzburg	brussels	11 June 2000	881
7	AT	BE	vienna	brussels	15 August 2000	75
8	AT	BE	graz	gent	18 September 2000	666
9	AT	BE	oth AT	gent	02 February 2000	223
10	AT	BE	salzburg	gent	09 July 2000	830
11	AT	BE	vienna	gent	06 June 2000	938
12	AT	BE	graz	liege	10 July 2000	534
13	AT	BE	oth AT	liege	07 September 2000	871
14	AT	BE	salzburg	liege	01 May 2000	928
15	AT	BE	vienna	liege	23 February 2000	129
16	AT	BE	graz	oth BE	28 September 2000	316
17	AT	BE	oth AT	oth BE	06 December 2000	583
18	AT	BE	salzburg	oth BE	04 January 2000	586
19	AT	BE	vienna	oth BE	27 August 2000	97
20	AT	CH	graz	basel	08 November 2000	479
21	AT	CH	oth AT	basel	29 April 2000	772
22	AT	CH	salzburg	basel	23 September 2000	385
23	AT	CH	vienna	basel	24 October 2000	624
24	AT	CH	graz	geneve	25 February 2000	165
25	AT	CH	oth AT	geneve	24 June 2000	776
26	AT	CH	salzburg	geneve	31 January 2000	151
27	AT	CH	vienna	geneve	06 January 2000	260
28	AT	CH	graz	lugano	10 March 2000	203
29	AT	CH	oth AT	lugano	08 July 2000	78
30	AT	CH	salzburg	lugano	26 August 2000	516
31	AT	CH	vienna	lugano	29 September 2000	250
32	AT	CH	graz	oth CH	03 June 2000	310
33	AT	CH	oth AT	oth CH	26 June 2000	694

Maximum value where Month is Feb or June and toCity starts with "Oth"

```
{=MAX(FILTER.IFS(0,$A$3:$F$10303,"Value",MONTH($E$3:$E$10303),{2,6},"toCity","~oth*"))}
```

997

List of alternate criteria values

Total value where Month is Feb or June and toCity starts with "Oth"

```
{=ASUMIFS(0,CountryCityData,"Value",MONTH($E$3:$E$10303),{2,6},"toCity","~oth*")}
```

189,952

Show all the rows where FromCtry=AT, month is June, day of the month is >20 and the toCity starts with "Oth"

```
{=PAD.ARRAY("",FILTER.IFS(1.CountryCityData,0,"FromCtry","AT",MONTH($E$3:$E$10303),6,DAY($E$3:$E$10303),">20","ToCity","~Oth*"))}
```

AT	CH	oth AT	oth CH	26 June 2000	694
AT	HU	graz	oth HU	21 June 2000	555
AT	PL	oth AT	oth PL	27 June 2000	408
AT	PT	salzburg	oth PT	28 June 2000	239

The first Criteria column FromCtry is sorted

0 means return ALL columns

Use either the column label or the relative column number

Create a calculated column by using an expression.

~ is the Like operator: and * means any character or characters

The formula is entered as a multi-column, multi-row array formula by selecting the block of cells, typeing the formula and then pressing Control-Shift-Enter. Excel shows { and } before and after the formula to show that it recognised the array formula.
Wrapping the formula in PAD.ARRAY makes any cells that will not contain results show a blank ("")

Total value of any 2 dynamic parameters: change the column name or any of the 2 cells of criteria values for each criteria column

```
=ASUMIFS(0,CountryCityData,"Value",$J$32,J33:K33,$H$32,H33:I33)
```

ToCity		FromCtry		Value	
London		~*		58155	From all Countries to London
Berlin	London	AT	BE	8197	From AT or BE to Berlin or London
Paris	Rome	<GB	>SE	46614	To Paris or Rome from Alphabetically < GB or >SE

Sum the value where either fromCity starts with Oth, OR toCity starts with Oth, avoiding double counting

```
=ASUMIFS(0,CountryCityData,"Value","FromCity","~oth*","#or",0,"ToCity","~oth*")
```

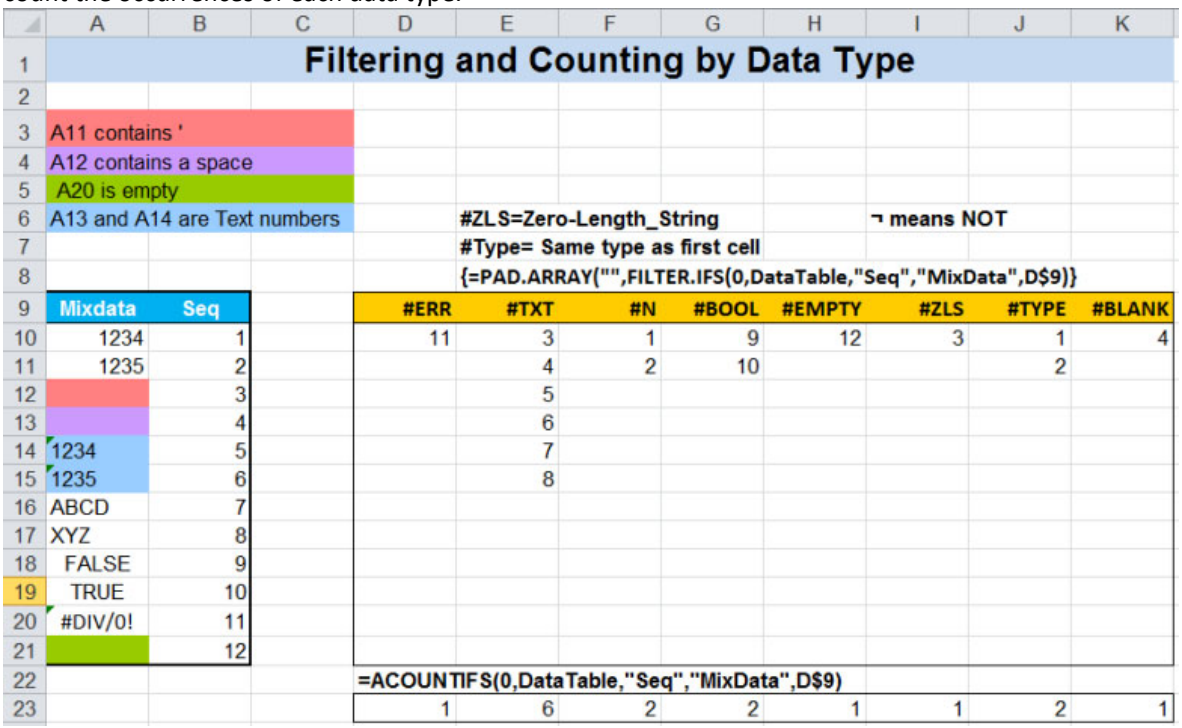
1935461

Sum the value where either fromCity starts with Oth, OR toCity starts with Oth, including double counting

```
=ASUMIFS(0,CountryCityData,"Value","FromCity","~oth*")+ASUMIFS(0,CountryCityData,"Value","ToCity","~oth*")
```

2142648

This example shows how to use FILTER.IFS to filter out data by data type, and how to use ACOUNTIFS to count the occurrences of each data type.



Filtering and Counting by Data Type

A11 contains '
A12 contains a space
A20 is empty
A13 and A14 are Text numbers

#ZLS=Zero-Length_String ¬ means NOT
#Type= Same type as first cell
{=PAD.ARRAY("",FILTER.IFS(0,DataTable,"Seq","MixData",D$9)}

Mixdata	Seq
1234	1
1235	2
	3
	4
1234	5
1235	6
ABCD	7
XYZ	8
FALSE	9
TRUE	10
#DIV/0!	11
	12

#ERR	#TXT	#N	#BOOL	#EMPTY	#ZLS	#TYPE	#BLANK
11	3	1	9	12	3	1	4
	4	2	10			2	
	5						
	6						
	7						
	8						

=ACOUNTIFS(0,DataTable,"Seq","MixData",D$9)

1	6	2	2	1	1	2	1

For the same data here are the results of using the ¬ (NOT) operator with data types.

¬#ERR	¬#TXT	¬#N	¬#BOOL	¬#EMPTY	¬#ZLS	¬#TYPE	¬#BLANK
1	1	3	1	1	1	3	1
2	2	4	2	2	2	4	2
3	9	5	3	3	4	5	3
4	10	6	4	4	5	6	5
5	11	7	5	5	6	7	6
6	12	8	6	6	7	8	7
7		9	7	7	8	9	8
8		10	8	8	9	10	9
9		11	11	9	10	11	10
10		12	12	10	11	12	11
12				11	12		12
11	6	10	10	11	11	10	11

FILTER.SORTED function

Filter.Sorted filters out subsets of sorted data using multiple extended conditions.

FILTER.SORTED is a simpler way of using FILTER.IFS when the first criteria is sorted.

FILTER.SORTED Syntax

FILTER.SORTED(RangeArray,ReturnCol,CriteriaColumn1,Criteria1, CriteriaColumn2,Criteria2, ..., ["#OR#", nSortedCrits,] CriteriaColumnx, Criteriax, ...)

See FILTER.IFS for definitions of the parameters for FILTER.SORTED

The RangeArray may optionally contain column labels in the first row, in which case at least one of the criteria columns or return column should be identified using a label. If you only use column numbers then the RangeArray should NOT include labels

If nSortedCrits is NOT equal to zero the RangeArray must be sorted on the criteria columns in the DataRange.

FILTER.MATCH function

FILTER.MATCH returns pairs of numbers (1-based relative position and number of rows) for each contiguous block of rows that meet the criteria.

The first column contains the relative position (first row is 1) within the RangeArray of the first row in the block that meets all the criteria. The second column contains the number of rows in the contiguous block.
The number of pairs returned by FILTER.MATCH is dependent on the data and the criteria being used. The number pairs can be used by OFFSET to return the block of rows.

FILTER.MATCH Syntax

FILTER.MATCH(nSortedCrits, RangeArray, CriteriaColumn1, Criteria1, CriteriaColumn2, Criteria2, …, ["#OR#", nSortedCrits,] CriteriaColumnx, Criteriax, …)

See FILTER.IFS for definitions of the parameters for FILTER.MATCH

FILTER.MATCH Example

Use FILTER.MATCH to find the start index and number of rows for oth CH

{=PAD.ARRAY(0,FILTER.MATCH(0,CountryCityData,"toCity","toCity",H49))}

Criteria	Start Row	Row Count
oth CH	30	4
	434	4
	1343	3
	1748	12
	2798	6
	3392	6
	3942	2
	4236	10
	5153	7
	5795	3
	6090	2
	6296	2
	6623	13
	7742	6
	8303	3
	8609	3
	8915	3
	9221	3
	9538	4
	9942	4
	0	0
	0	0

Use the Output from FILTER.MATCH

=IF(I49>0,INDEX(CountryCityData,I49,1),"")

=IF(J49>0,SUM(OFFSET(CountryCityData,I49-1,5,J49,1)),"")

FromCtry	Total Value
AT	1923
BE	1008
CZ	1811
DE	7816
DK	3243
ES	3135
FI	1480
FR	3896
GB	3750
GR	1100
HU	853
IE	763
IT	6112
NL	2360
NO	793
PL	2276
PT	1507
RU	1254
SE	1886
TR	11400

ASUMIFS function

ASUMIFS sums the cells in the chosen column(s) that match the extended conditions or criteria.

ASUMIFS Syntax

ASUMIFS (nSortedCrits, RangeArray, ReturnCol, CriteriaColumn1, Criteria1, CriteriaColumn2, Criteria2, ..., ["#OR#", nSortedCrits,] CriteriaColumnx, Criteriax, ...)

ASUMIFS() is equivalent to SUM(FILTER.IFS())

See FILTER.IFS for definitions of the parameters for ASUMIFS.

The RangeArray may optionally contain column labels in the first row, in which case at least one of the criteria columns or return column should be identified using a label. If you only use column numbers then the RangeArray should NOT include labels,

If nSortedCrits is NOT equal to zero the RangeArray must be sorted on the criteria columns in the DataRange.

If ReturnCol is zero then a row of sums will be spilled for each column in RangeArray. This is an efficient method of summing multiple columns for a common set of criteria.

ACOUNTIFS function

ACOUNTIFS counts the cells in the chosen column(s) that match the extended conditions or criteria.

ACOUNTIFS Syntax

ACOUNTIFS (nSortedCrits, RangeArray, ReturnCol, CriteriaColumn1, Criteria1, CriteriaColumn2, Criteria2, ..., ["#OR#", nSortedCrits,] CriteriaColumnx, Criteriax, ...)

ACOUNTIFS() should always be used rather than COUNTA(FILTER.IFS()) because COUNTA counts a return of zero as 1 but ACOUNTIFS does not.

See FILTER.IFS for definitions of the parameters for ACOUNTIFS

The RangeArray may optionally contain column labels in the first row, in which case at least one of the criteria columns or return column should be identified using a label. If you only use column numbers then the RangeArray should NOT include labels

If nSortedCrits is NOT equal to zero the RangeArray must be sorted on the criteria columns in the DataRange.

If ReturnCol is zero then a row of counts will be spilled for each column in RangeArray. This is an efficient method of counting multiple columns for a common set of criteria.

FILTER.VISIBLE function

FILTER.VISIBLE returns an array containing all the visible rows in the input Range. Rows that are hidden, have zero height or are filtered out by AutoFilter are excluded. **The input data must be a range**.
FILTER.VISIBLE is Volatile, is NOT multithreaded and is an array function.

FILTER.VISIBLE Syntax

FILTER.VISIBLE(theRange, Pad)

The first parameter is required, the second is optional.

theRange (Required)

A range reference for the rows and columns to be filtered.

Pad (Optional default #N/A)

The value to use for any excess cells in the array formula. Excess cells are cells in a CSE entered multi-cell array formula that have no corresponding output from the function.
If FILTER.VISIBLE is called from a multi-cell CSE entered array formula it will return the smaller of the number of rows in the multi-cell array formula or the number of rows in the input range.
If FILTER.VISIBLE is called from a single-cell CSE entered array formula it will return the number of rows in the input range.

Rgx.COUNTIF function

Counts the number of values within a range or array that match the Regular Expression Pattern.

Rgx.COUNTIF as a multi-threaded non-volatile function.

Rgx.COUNTIF Syntax

Rgx.COUNTIF(SearchThis, RegExp, Case_Sensitive)

The first 2 parameters are required; the last parameter is optional.

SearchThis (required)

A rectangular range, array or expression that returns a rectangular array to be searched for matches to the Regular Expression Pattern. All rows and columns within the rectangular array are evaluated for a match.

RegExp (required)

Specifies the Regular Expression to be used when matching values from SearchThis

Case_Sensitive (optional)

TRUE to make the pattern-matching case-sensitive. The default is FALSE.

Rgx.COUNTIF Example

Numbers	RegEx Pattern
1-234-567-8901	1?\W*([2-9][0-8][0-9])\W*([2-9][0-9]{2})\W*([0-9]{4})(\se?x?t?(\d*))?
1-234-567-8901 x1234	
1-234-567-8901 ext1234	**Rgx.COUNTIF($A$36:$A$42,$C$35)**
1 (234) 567-8901	
1.234.567.8901	6
1/234/567/8901	
12345678901	

The Numbers data is in A36:A42 and the Regular Expression Pattern is in C35. There are 6 numbers that match the pattern. The Numbers data is all textual numbers.

Rgx.SUMIF function

Counts the number of values within a range or array that match the Regular Expression Pattern. Rgx.SUMIF is a multi-threaded non-volatile function.

Rgx.SUMIF Syntax

Rgx.SUMIF(Search_This, RegExp, Sum_This, Case_Sensitive)

The first 2 parameters are required; the last parameter is optional.

Search_This (required)

A rectangular range, array or expression that returns a rectangular array to be searched for matches to the Regular Expression Pattern. All rows and columns within the rectangular array are evaluated for a match.

RegExp (required)

Specifies the Regular Expression to be used when matching values from SearchThis

Sum_This (required)

A rectangular range, array or expression that returns a rectangular array that is the same size as Search_This.

Each of the values in positions in Sum_This that correspond to values matched in Search_This will be SUMmed.

Case_Sensitive (optional)

TRUE to make the pattern-matching case-sensitive. The default is FALSE.

Rgx.SUMIF Example

Numbers	Values	RegEx Pattern
1-234-567-8901	43	1?\W*([2-9][0-8][0-9])\W*([2-9][0-9]{2})\W*([0-9]{4})(\se?x?t?(\d*))?
1-234-567-8901 x1234	16	
1-234-567-8901 ext1234	10	**Rgx.SUMIF($A$36:$A$42,$C$36,$B$36:$B$42)**
1 (234) 567-8901	86	
1.234.567.8901	31	264
1/234/567/8901	78	
12345678901	22	

The Numbers data is in A36:A42 and the Regular Expression Pattern is in C35. The Values to be summed are in B36:B42.

There are 6 numbers that match the pattern, and the corresponding values are summed to give 264. The Numbers data is all textual numbers

FILTERX function

An extended version of FILTER that allows the extended criteria from the FILTER.IFS family.

- FILTERX is a multi-threaded non-volatile dynamic array function.
- FILTERX handles successive criteria progressively – each criteria only processes rows that pass the previous criteria test.
- When the first criteria column is sorted (FiirstCritSort=1 or -1) FILTERX uses binary search to efficiently do the search.
- FILTERX columns can be ranges, calculated columns or arrays.
- Up to 18 pairs of Criteria and Columns may be used.
- 3 Types of Criteria Lists: Include, Exclude and All-Of

FILTERX Syntax

FILTERX([FirstCritSort,] Array, Col1, Crit1 [, Col2, Crit2, …])

FirstCritSort (Optional)

If omitted the first Criteria column is assumed to be unsorted (and the trailing comma is not required)
A value of 1 indicates that Col1 is sorted ascending, -1 indicates sorted descending.
When sorted binary search will be used for efficient criteria matching on the first criteria column.

Array (required)

The range or array to be filtered. A complete row from Array will be returned for row that meets all of the criteria.

Col1 (required)

Specifies the first Criteria Column. Can be text, a number, a Range Reference, an array of constants or an expression that results in a column array.
Text will be interpreted as a column label in the first row of the RangeArray
Number will be interpreted as a column number within RangeArray.
Range Reference or Array will be interpreted as an independent column outside RangeArray.
An array of constants will be interpreted as a column of data.
An expression will be evaluated by Excel as a calculated column before being passed to the FILTER function.
Criteria Columns must contain the same number of rows as the RangeArray, but allowance is made for a row of column labels in either or both the RangeArray and Criteria Columns.

Crit1 (required)

Specifies the criteria expression to be applied to CriteraColumn1.
The Criteria expression must resolve to a text string starting with an optional criteria operator and ending with a Criteria value.
If no criteria operator is found at the start then = will be assumed.
A Criteria expression may be any Excel expression containing strings, range references, defined names, functions and operators

The maximum number of pairs of Criteria Columns and their criteria is 18.

Col2 (optional)

The column (range, array or calculated column) to be searched using Crit2

Crit2 (optional)

The criteria to be applied to Col1

Criteria Operators

Valid Criteria operators are:

The relational operators:

- = is equal to
- > is greater than
- >= is greater than or equal to
- < is less than
- <= is less than or equal to
- ¬ is not equal to, also negates following criteria operators
- If no operator is given it is treated as =, except that the data type is never converted (see Data Types section below).

The Pattern operators:

- ~ Like the criteria value pattern using the wild-card characters * and ?
- ~~ Matches against a regular expression pattern

The Boolean operators:

- True the cell value is true
- False the cell value is false

Because Excel propagates error values through formulas and expressions all the above criteria operators have no effect when preceding an error value.

Criteria Lists

Criteria can be given either as a single criterion or a list of alternative Criteria.

The list of alternative Criteria can be either an expression returning an array, an array of Constants or a Range Reference.

The list of alternative criteria can either be:

- An **Inclusion list** – a row that matches any of the items in the list will be included.
- An **exclusion list** – a row that matches any of the items in the list will be excluded. The first element in the list must be two not symbols ¬¬ entered as a string.
- An **All list** – row will must match All of the items in the list to be included. The first element in the list must be && (two & characters) entered as a string. This can be used for example to match between 2 dates.

The individual elements in the array or cells in the Range Reference may also each start with their individual criteria Operator, or may not have a criteria operator.

Criteria Lists Examples:

FILTER.IFS(0,$A$1:$C$6,"Hundreds","Digits",{"Four","One","Two"},"Tens",{"Fifty","Fourty","Thirty","Twenty"})

The Digits column must contain any of Four, One or Two and the Tens column must contain any of Fifty, Forty, Thirty or Twenty.

FILTER.IFS(0,$A$1:$C$6,"Hundreds","Digits",{"Four","One","Two"},"Tens",{"¬¬","Fifty","Fourty","Thirty","Twenty"})

The Digits column must contain any of Four, One or Two and the Tens column must NOT contain any of Fifty, Forty, Thirty or Twenty.

FILTERX(tblSalesData,tblSalesData[ShipDate],VSTACK("&&", ">=" & Date(2026,9,24), "<=" & Date(2026,12,31)))

The Sales Data table will be filtered for sales with ship dates between 24 Sep 2026 and 31 Dec 2026 inclusive.

For very large lists it will be more efficient to use COMPARE.LISTS:

FILTERX(tblSalesData,tblSalesData[ShipDate],VSTACK("&&", ">=" & Date(2026,9,24), "<=" & Date(2026,12,31)),COMPARE.LISTS(tblSalesData[Products),HighValueProducts],True)

The Sales Data table will be filtered for the list of High Value Products with ship dates between 24 Sep 2026 and 31 Dec 2026 inclusive.

The Data Type Operators:

- #TYPE the data type of the cell is the same as the first data cell in the criteria column
- #ERR the cell contains an error value
- #TXT the cell contains a string (text) value
- #N the cell contains a number (integer or with decimal point, date, time, currency)
- #BOOL the cell contains a Boolean True or False
- #EMPTY the cell is empty
- #ZLS the cell contains a zero length string
- #BLANK the cell contains a string of 1 or more blanks/spaces

Any of these criteria operators can be prefixed by ¬ (not) to convert the operator from an inclusion operator to an exclusion operator.

Data Type Comparison

Excel has 5 fundamental data types: Number (integers, real numbers, dates, times, currency), String (text), Boolean (true or false), Error (#NA etc) and Empty (unused cells).
When a column of mixed data types is sorted Excel uses the following comparison relationships:
Numbers < Strings < Booleans < Errors
Empty cells are always sorted last, both in an ascending and a descending sort.
The SpeedTools FILTER functions use the same comparison relationship between data types.

Data Type Conversion

When a criteria value is preceded by any of the criteria operators its data type is not known (Criteria values without any preceding criteria operator have a specific data type).
For example, in "<1234" the 1234 could be a number or could be a string. When this criteria value is being compared to a criteria column that could contain multiple data types the FILTER.IFS functions use the following rules:
When the criteria operator is ~ or ~~ the criteria value and the values in the criteria column will be converted to strings before doing the pattern match.
Otherwise **the preferred data type will be the data type of the first data (non-header) cell in the criteria column**. FILTER.IFS will attempt to convert the criteria value to the preferred data type. If this is not possible FILTER.IFS will convert the criteria value to a number, then a string, then a Boolean then an error.

Overriding Data Type Conversion using the & Prefix

If your criteria column contains mixed data types (for instance both numeric and string numbers) you can ask FILTER.IFS to try to convert the criteria value to the best possible data type match with each cell in the criteria column.
So FILTER.IFS would use the numeric version of the criteria value to compare with a numeric number and the string version of the criteria value to compare with a string/text number.
You request this by adding a & character as a prefix to the criteria operator (after the ¬ prefix if you are using the exclusion prefix).

Criteria Values

Criteria values and Criteria column values are NOT Case sensitive.

Criteria Like Patterns

Like Patterns can contain:

- * Any number of characters, including none
- ? Any single character

Pattern Examples:

Column Value	Criteria	Result	Criteria Explanation
"aBBBa"	"~a*a"	True	String starting with a followed by any characters and ending with a
"a2a"	"~a?a"	True	String starting with a followed by any single character followed by a
"BAT123khg"	"~B?T*"	True	B followed by any single character followed by T followed by any or no characters
"CAT123khg"	"~B?T*"	False	B followed by any single character followed by T followed by any or no characters

CriteriaColumn2, Criteria2, … (optional)

You can give additional pairs of Criteria Column and Criteria.

All of the Criteria pairs given in a set must be True for a row of data from the Return Column to be included.

"#OR#", nSortedCriteria, (optional)

If you need alternative sets of criteria you can separate them with "#OR#", nSortedCrits.

A row will be selected from the Return Column if ANY of the alternative sets of Criteria are met.

The row will only be selected once if both alternative sets of criteria are met (no double-counting).

For Example:

CriteriaCol1,Criteria1,CriteriaCol2, Criteria2, "#OR#",1,CriteriaCol3, Criteria3,CriteriaCol4,Criteria4

A row will be selected from the Return Column if

(both criteria1 and Criteria2 are True) OR (both Criteria3 and Criteria4 are True)

Note that a Criterion can also contain a list of alternative conditions (see above)

The SPAN family of functions

The SPAN family of functions provide an easy and efficient way of working with a span of data between two values.

MATCH.SPAN and **SPAN** are frequently used together, where MATCH.SPAN is used to find the pair of row or column indexes that include the span, and SPAN is used to extract the span using the pairs supplied by MATCH.SPAN.

INSPAN- Is primarily used in FILTER, FILTERX and the *IFS family of functions. It returns an array of True/False for a span defined by a From_Value and a To_Value in a single row or column of an array or range.

MATCH.SPAN Function

Returns an array containing 2 values giving the position of two inclusive search values within a single-column or single-row range or array. The returned array will be a 2 value row for a column or a 2 value column for a row.
MATCH.SPAN is a multi-threaded non-volatile dynamic array function.

MATCH.SPAN Syntax

MATCH.SPAN(Range_Array,From_Value, To_Value, Match_Mode)

Range_Array (Required)

A single column or row of an array or range to be searched.

From_Value (Required)

The value to be searched for, starting at the top of Range_Array and searching down the rows or across the columns. For duplicated values the first value found is used.

To_Value (Required)

The value to be searched for, starting at the bottom/right of Range_Array and searching up the rows or right-to-left in the columns. For duplicated values the first value found is used.

Note: From_Value must always be less than To_Value even when the Range_Array is sorted descending.

Match_Mode (Optional – default 0 unsorted)

0 Unsorted – Linear search method is used
1 Sorted Ascending – very efficient binary search method is used
-1 Sorted Descending – very efficient binary search method is used.

Handling of errors and out of bounds conditions

When Range_Array is sorted (Match_Mode 1 or -1)

Out of Bounds:

From and To values outside the first or last value in Range_Array are treated as the corresponding first or last value, unless both the From and To values are less than the first value or greater than the last value (there is no span so error).

Missing Values:

When From or To values are missing within Range_Array the next closest value within the span is used.

Duplicate Values:

When there are duplicated From and To values within Range_Array the outermost value found is used.

Empty Span:

From and To values that produce an empty span are treated as errors (#N/A).

When Range_Array is unsorted (Match_Mode 0)

Out of Bounds:

From and To values outside the first or last value in Range_Array are treated as errors (#N/A).

Missing Values:

Missing From or To values are treated as errors (#N/A)

When there are duplicated From and To values within Range_Array the outermost value found is used.

Empty Span:

From and To values that produce an empty span are treated as errors (#N/A).

SPAN Function

Extracts a span of rows and columns defined by a pair of row numbers and a pair of column numbers from a range/array.
SPAN is a multi-threaded non-volatile dynamic array function.

SPAN Syntax

SPAN(Range_Array, [Row_Pair], [Column_Pair])

Range_Array (required)

The range or array to be used as the source for the extraction.
If Range_Array is a Range reference then SPAN will return a Range, otherwise SPAN will return an array.

Row_Pair (Optional)

A pair of values given as a range or array that defines the start and end rows within Range_Array to extract. If omitted all the rows in Range_Array will be used.

Column_Pair (Optional)

A pair of values given as a range or array that defines the start and end columns within Range_Array to extract. If omitted all the columns in Range_Array will be used.

Handling of errors and out of bounds conditions

If any of the pair values exceed the bounds of Range_Array an error will be returned.

INSPAN Function

Returns True for the position of all values between and including two search values, otherwise False. INSPAN is a multi-threaded non-volatile dynamic array function.

INSPAN Syntax

INSPAN(Range_Array,From_Value, To_Value, Match_Mode)

Range_Array (Required)

A single column or row of an array or range to be searched.

From_Value (Required)

The value to be searched for, starting at the top of Range_Array and searching down the rows or across the columns. For duplicated values the first value found is used. This value is included in the results.
If From_Value does not exist in or is beyond Range_Array the next nearest value to To_Value is used.

To_Value (Required)

The value to be searched for, starting at the bottom/right of Range_Array and searching up the rows or right-to-left in the columns. For duplicated values the first value found is used. This value is included in the results.
If From_Value does not exist in or is beyond Range_Array the next nearest value to From_Value is used.

Note: From_Value must always be less than To_Value even when the Range_Array is sorted descending.

Match_Mode (Optional – default 0 unsorted)

0 Unsorted – Linear search method is used
1 Sorted Ascending – very efficient binary search method is used
-1 Sorted Descending – very efficient binary search method is used.

Handling of errors and out of bounds conditions

When Range_Array is sorted (Match_Mode 1 or -1)
Out of Bounds:
From and To values outside the first or last value in Range_Array are treated as the corresponding first or last value, unless both the From and To values are less than the first value or greater than the last value (there is no span so error).
Missing Values:
When From or To values are missing within Range_Array the next closest value within the span is used.
Duplicate Values:
When there are duplicated From and To values within Range_Array the outermost value found is used.
Empty Span:
From and To values that produce an empty span are treated as errors (#N/A).

When Range_Array is unsorted (Match_Mode 0)
Out of Bounds:
From and To values outside the first or last value in Range_Array are treated as errors (#N/A).
Missing Values:
Missing From or To values are treated as errors (#N/A)
When there are duplicated From and To values within Range_Array the outermost value found is used.
Empty Span:
From and To values that produce an empty span are treated as errors (#N/A).

Span Function Examples

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Sorted Values with**												
2	**Duplicates and Missing Values**												
3	11										=INSPAN($A$3:$A$21,C7,D7,1)		
4	12											FALSE	
5	13		From	To		=MATCH.SPAN($A$3:$A$21,C6,D6,1)						FALSE	
6	14		10	30		1	19	Outside boundaries				FALSE	
7	15		19	21		9	12	Duplicate to Missing				FALSE	
8	16		19	19		9	12	Duplicate to Duplicate				FALSE	
9	17		20	25		13	15	Missing to Missing				FALSE	
10	18		7	10		#N/A	#N/A	Both before				FALSE	
11	19											FALSE	
12	19					=SPAN($A$3:$A$21,F9#)						TRUE	
13	19					23						TRUE	
14	19					23						TRUE	
15	23					23						TRUE	
16	23											FALSE	
17	23					=SPAN($A$3:$A$21,MATCH.SPAN($A$3:$A$21,C7,D7,1))						FALSE	
18	26					19						FALSE	
19	27					19						FALSE	
20	28					19						FALSE	
21	29					19						FALSE	
22												FALSE	

Lookup and Match Functions

An efficient, advanced and powerful set of Lookup functions.

Outstanding Performance

SpeedTools Lookups outperforms Excel's Lookup functions by using better algorithms and efficient C++ multi-threaded coding:

- Exact Match Binary Search
- Memory Lookups
- Internal Hash tables for List comparisons and array lookups

Advanced Function

The table below shows a summary comparison of the features in Excel and SpeedTools Lookup functions.

Feature	Excel VLOOKUP	MEMLOOKUP	AVLOOKUP family
Sort Default	Ascending	Not Sorted	Not Sorted
Default Match Type	Approx	Exact	Exact
Linear Search	Yes	Yes	Yes
Approx Binary Search	Yes	No	Yes
Exact Binary Search	No	Yes	Yes
Re-usable Lookup Memory (Row)	No	Yes	Yes
Re-usable Lookup Memory (Col)	No	Yes	Yes
Re-usable Lookup Memory (Cell)	No	Yes	Yes
Multi-Threaded	Yes	Yes	Yes
Horizontal Lookup	HLOOKUP	Option	Vertical only
Array, Expression or Range	Yes	Yes	Yes
Lookup Any Column	No	No	Yes
Use Column Labels or Numbers	No	No	Yes
Multiple Lookup Columns (AND)	No	No	Yes
Multiple Lookup Rows (OR)	No	No	Yes
Multiple Answer Columns	No	No	Yes
Return First, Last or Nth Match	No	No	AVLOOKUPNTH
2 Dimensional Lookup	INDEX(…,…))	INDEX(…,…)	Yes
Return All Matches	No	No	AVLOOKUPS2
No Match Error-handling	IFERROR()	IFERROR()	Built-in option
Compare Lists	V Slow	Faster	Super-Fast (COMPARE.LISTS)
Wild Card Text Match	Yes	Yes	Yes
Case-Sensitive Match	No	No	Case.AVLOOKUP2
Regular Expression Text Match	No	No	Rgx.AVLOOKUP2…

Better, Safer Lookup Defaults

All the SpeedTools lookup and match functions default to unsorted exact match, since that is the safest option. Excel's VLOOKUP and MATCH default to sorted approximate match, which is likely to give incorrect answers in most cases!

SpeedTools Lookup Families

The 28 Lookup functions are organized into families to simplify choosing the particular function best suited to your needs.

- MEMLOOKUP - Fast exact match Lookup using memory
- MEMMATCH - Fast exact match MATCH using memory
- AVLOOKUP2 - Fast powerful advanced function lookup
- AVLOOKUPS2 - Advanced function lookup returning ALL matches
- AVLOOKUPNTH - Advanced function lookup returning the Nth match
- AMATCH2 - Fast powerful advanced function MATCH
- AMATCHES2 - Advanced function MATCH returning ALL matches
- AMATCHNTH - Advanced function MATCH returning the Nth match
- Case.AVLOOKUP2 - Case-sensitive version of AVLOOKUP2
- Case.AVLOOKUPNTH - Case-sensitive version of AVLOOKUPNTH
- Case.AVLOOKUPS2 - Case-sensitive version of AVLOOKUPS2
- Case.AMATCH2 - Case-sensitive version of AMATCH2
- Case.AMATCHES2 - Case-sensitive version of AMATCHES2
- Case.AMATCHNTH - Case-sensitive version of AMATCHNTH
- Rgx.AVLOOKUP2 - Regular Expression version of AVLOOKUP2
- Rgx.AVLOOKUPNTH - Regular Expression version of AVLOOKUPNTH
- Rgx.AVLOOKUPS2 - Regular Expression version of AVLOOKUPS2
- Rgx.AMATCH2 - Regular Expression version of AMATCH2
- Rgx.AMATCHES2 - Regular Expression version of AMATCHES2
- Rgx.AMATCHNTH - Regular Expression version of AMATCHNTH
- Rgx.Case.AVLOOKUP2 - Case-sensitive version of Rgx.AVLOOKUP2
- Rgx.Case.AVLOOKUPNTH - Case-sensitive version of Rgx.AVLOOKUPNTH
- Rgx.Case.AVLOOKUPS2 - Case-sensitive version of Rgx.AVLOOKUPS2
- Rgx.Case.AMATCH2 - Case-sensitive version of Rgx.AMATCH2
- Rgx.Case.AMATCHES2 - Case-sensitive version of Rgx.AMATCHES2
- Rgx.Case.AMATCHNTH - Case-sensitive version of Rgx.AMATCHNTH
- EVAL2 - Evaluate a formula string

High-performance exact match Memory Lookups

Written in C++, SpeedTools multi-threaded Lookup functions provide faster and easy-to-use alternatives to Excel's MATCH and VLOOKUP functions **for exact match lookups on both sorted and unsorted data.**

Excel VLOOKUP does exact match lookups by starting at the first row and looking at each row in turn until a match is found. For large numbers of rows this **linear search is slow**.
SpeedTools Lookups remember which rows gave a match the last time they were calculated and will try to **short-circuit the slow linear search** by checking the remembered row first. Because Excel has to recalculate Lookups whenever any of the values in the Lookup Table change (even when the answer does not change) this technique can provide significant speedups.

For sorted data SpeedTools Lookups uses exact **match binary search** which is much faster than the exact match linear search you have to use with Excel lookups.

Lookup Memory is Fail-Safe

If the memory row does NOT give an exact match to the value being looked up then SpeedTools Lookups will do an exact match linear search or exact match binary search.

Four different kinds of Lookup Memory

SpeedTools Lookups can use 4 different kinds of multi-threaded memory:

Re-useable Global Memory for Rows or Columns

A single index is stored in global memory for each row (vertical lookups) or column (horizontal lookups). Global memory is super-efficient and easy to use because it gets re-used when you have more than one lookup formula on the same row looking up different columns in a Table. This global memory works across all worksheets in all open workbooks.

Book Sheet Row Memory (default option)

If you are using different Lookup Tables on different worksheets but still want to have more than one lookup formula on a row looking up different columns from the same table you should use this option which stores the row memory separately for each workbook & worksheet.

Book Sheet Cell Memory

If you are using multiple Lookup Tables formulas on the same sheet and the same row but in different cells you can use this option which stores the memory separately for each cell.

Named Row Memory within Workbook

If you are using a number of Lookup Tables it may be better to use a separate Named Row Memory for each table. This allows you to use multiple memory lookup formulas that look up the tables in the same row or even in the same cell. The named row memory is shared by name across all worksheets within a workbook.

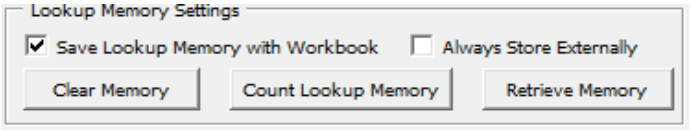
Lookup Memory is stored with the workbook

All four kinds of Lookup Memory can be stored with the workbook so that it is automatically restored when you next open the workbook.

Managing the Global Lookup Memory pools

The current Lookup Memory is stored each time the workbook is saved. When a workbook is opened its stored Lookup Memory is added to the global Lookup Memory pool.
The Lookup Memory Pool persists until the Excel session is closed or Lookup Memory is cleared.

The SpeedTools Settings tab in FastExcel SpeedTools Calculation Options and Settings enables you to control Lookup Memory.



You can choose whether to save the Lookup Memory and whether to always store it externally to the workbook. For Excel 2003 and earlier the Lookup Memory is always stored externally. External memory files are stored in the same directory as the workbook and have file extensions of **.mrl**, **.mcl** and **.mcs**

Clear Memory empties the current Lookup Memory.

Count Lookup Memory gives you a count of each of the three different kinds of Lookup Memory.

Retrieve Memory repopulates the Lookup Memory from the workbook or the external files.

Choice of simple or advanced Lookup Functions

MEMMATCH and MEMLOOKUP provide a simple way of replacing your existing exact match VLOOKUP, HLOOKUP, LOOKUP and MATCH formulas with more efficient memory-based functions.

MEMMATCH and MEMLOOKUP can accept arrays and calculated ranges for the Lookup_Table, and will do vertical or horizontal lookups depending on the shape of the Lookup Table.

There an additional 24 Advanced SpeedTools Lookup functions that give you many extended options.

The 24 Advanced Function Lookups

If you need more powerful functions use these advanced Lookup features:

- High-performance multi-threaded lookups to exploit multiple cores
- Written in C++ for ultimate performance
- Run with 32 or 64 bit Excel
- 3 multi-threaded memory options
- Return the first, last or Nth match from multiple answers (does not use lookup memory)
- Return all matching answers where duplicates exist in the Lookup Columns to either multi-cell array formulas or aggregating functions like SUM, AVERAGE, MEDIAN, COUNT etc.
- Two-dimensional Lookup: lookup both rows and columns
- Multi-column Lookup (AND): lookup a row based on more than one column.
- Multi-row Lookup (OR): Allows for multiple rows of lookup values and multiple lookup answers. Use multi-row Lookups in array formulas or in formulas like SUM, AVERAGE, MEDIAN, COUNT
- Use multiple answer columns to return values from multiple columns.
- The lookup column(s) can be any column(s) in the Lookup_Table
- Specify the Lookup Column(s) and Answer Column(s) using Column header labels or column numbers.
- Choice of case-sensitive or not case-sensitive matching
- Lookups can use regular expressions for text exact match lookup
- Lookup_Table can be sorted ascending, descending or not sorted
- Lookup_Table can be a range or an array or a calculated range or an expression yielding an array.
- Separate Sort and Exact options allow Exact Match with sorted data
- Specify the error returned when Exact Match cannot find a matching row
- Use wildcard characters * and # in text exact match lookup.

The Advanced Lookups Flavors

The advanced LOOKUP functions are combined in a number of different flavors.

- AVLOOKUP2 and AMATCH2
- Return all matches; return the Nth match and the standard match return.
    - For example AMATCHES2, AMATCHNTH, AMATCH2
- Case-sensitive and non-case-sensitive
    - For example Case.AVLOOKUP2, Case.AVLOOKUPNTH, Case.AVLOOKUPS2
- Regular Expression lookup and standard lookup
    - For example Rgx.Case.AMATCH2, Rgx.AVLOOKUPS2, Rgx.AMATCHNTH

All combinations of these flavours make 24 different advanced lookup functions in addition to MEMLOOKUP, MEMMATCH and COMPARE.LISTS.

MEMLOOKUP Function

The MEMLOOKUP function uses Memory Lookup for faster exact match lookup on both sorted and unsorted data. Use MEMLOOKUP to replace VLOOKUP, HLOOKUP and LOOKUP.

MEMLOOKUP always does an exact match. On sorted data MEMLOOKUP does an exact match binary search.
If no match is found MEMLOOKUP returns #N/A even with sorted data.

For a vertical range or array MEMLOOKUP looks for a value in the leftmost column of Lookup_Array and then returns a value in the row where the value was found from a column you specify.
For a horizontal range or array MEMLOOKUP looks for a value in the topmost row of Lookup_Array and then returns a value in the column where the value was found from a row you specify.

MEMLOOKUP is a multi-threaded, non-volatile non-array function.

MEMLOOKUP Syntax

MEMLOOKUP (Lookup_Value, Lookup_Array, Col_index_num, Sort_Type, MemType_Name, Vertical_Horizontal)

The first 3 parameters are required; the last 4 parameters are optional.

Lookup_Value (required)

Specifies the value to be found in the Lookup_Array. Can be a constant or a range reference or an expression returning a single value.

Lookup_Array (required)

A vertical or horizontal array of constants, or a range reference or expression, that returns a contiguous rectangular table of Lookup Values.

Result_Column (required)

For a vertical Lookup_Array the column number or label in the header row in Lookup_Array from which the matching value should be returned. The first column in Lookup_array is column 1.
For a horizontal Lookup_Array the row number or label in the header column in Lookup_Array from which the matching value should be returned. The first row in Lookup_array is row 1.

If Result_Column is a string then Lookup_Array is assumed to contain a header row/column to be searched for Result_Column. If Result_Column cannot be found the function returns an error message "#Result Column not found in Header". If Result_Column is a number then Lookup_Array is assumed NOT to contain a header row/column.

Sort_Type (optional - default 0)

A number 1, 0 or -1 indicating the sort sequence of Lookup_Array.

- -1 Sorted Descending
- 0 Not Sorted
- 1 Sorted Ascending

The default is 0: not sorted

MemType_Name (Optional, Defaults to 2)

Memory type for lookups can be 0, 1, 2 or 3 or a string that is used as the name for a memory.

- **3 = Global Memory for Rows or Columns** A single index is stored in global memory for each row (vertical lookups) or column (horizontal lookups).
  This global memory is super-efficient and easy to use when you have more than one lookup formula on the same row looking up different columns in a Table. This global memory works across all worksheets in all open workbooks.
- **2 = Book Sheet Row Memory** If you are using different Lookup Tables on different worksheets but still want to have more than one lookup formula on a row looking up different columns in the same table you can use this option which stores the row memory separately for each Workbook worksheet.
- **1 = Book Sheet Cell Memory** If you are using multiple Lookup Tables formulas in different cells on the same sheet and the same row you should use this option which stores the memory separately for each cell.
- **0 = Do not use Lookup memory**
- **Named Memory** If you give a text string instead of a number it will be used as a named memory. You can have many named memories in use at the same time (usually one for each Lookup Table). Each named memory works by row across all sheets within a workbook, so is the best choice when you have:
  - More than one memory lookup referring to different lookup tables within a single formula.
  - More than one memory lookup referring to different lookup tables in different cells on the same row, and these lookups are repeated on the same row in multiple sheets.

If this option is not zero, then MEMLOOKUP will first check to see if the index stored in memory that gave the answer the last time the MEMLOOKUP was calculated still gives the correct answer. If it does then MEMLOOKUP will return that answer without doing any more processing.

When using Lookup memory and there are multiple exact match answers available MEMLOOKUP will not necessarily return the answer from the same row as VLOOKUP.

Vertical_Horizontal (Optional, Defaults to 1)

This parameter controls whether the lookup is done vertically or horizontally.

0 = Guess – if the number of rows >= the number of columns vertical, otherwise horizontal
1= Always Vertical (Default)
2= Always Horizontal

MEMLOOKUP Performance

If your data is **unsorted** and there is no Lookup Memory already available for the position of the MEMLOOKUP formula then MEMLOOKUP will do a linear search and store the row or column index found in the Lookup Memory for subsequent MEMLOOKUP calls.
In this case the second execution of the MEMLOOKUP formula will be much faster than the first.

If your data is **sorted** then MEMLOOKUP will do an **exact match Binary Search**, which is very fast, and the second execution of the MEMLOOKUP formula will also be very fast.

MEMMATCH Function

The MEMMATCH function uses Memory Lookup for faster exact match lookup on both sorted and unsorted data. Use MEMMATCH to replace the MATCH function.

MEMMATCH always does an exact match. On sorted data MEMMATCH does an exact match binary search.

If no match is found MEMMATCH returns #N/A even with sorted data.

For a vertical range or array MEMMATCH looks for a value in the leftmost column of Lookup_Array and then returns the relative position of the item in the Lookup_Array that matches the specified value (Lookup_Value).

For a horizontal range or array MEMMATCH looks for a value in the topmost row of Lookup_Array and then returns the relative position of the item in the Lookup_Array that matches the specified value (Lookup_Value).

MEMMATCH is a multi-threaded, non-volatile, non-array function.

MEMMATCH Syntax

MEMMATCH (Lookup_Value, Lookup_Array, Col_index_num, Sort_Type, MemType_Name, Vertical_Horizontal)

The first 3 parameters are required; the last 3 parameters are optional.

Lookup_Value (required)

The value to be found in the Lookup_Array. Lookup_Value can be a constant or a range reference or an expression returning a single value.

Lookup_Array (required)

A vertical or horizontal array of constants, or a range reference or expression, that returns a contiguous rectangular table of Lookup Values.

Sort_Type (optional - default 0)

A number 1, 0 or -1 indicating the sort sequence of Lookup_Array.

- -1 Sorted Descending
- 0 Not Sorted
- 1 Sorted Ascending

The default is 0: unsorted

MemType_Name (Optional, Defaults to 2)

Memory type for lookups can be 0, 1, 2 or 3 or a string that is used as the name for a memory.

- **3 = Global Memory for Rows or Columns** A single index is stored in global memory for each row (vertical lookups) or column (horizontal lookups).
  This global memory is super-efficient and easy to use when you have more than one lookup formula on the same row looking up different columns in a Table. This global memory works across all worksheets in all open workbooks.
- **2 = Book Sheet Row Memory** If you are using different Lookup Tables on different worksheets but still want to have more than one lookup formula on a row looking up different columns in the same table you can use this option which stores the row memory separately for each Workbook worksheet.
- **1 = Book Sheet Cell Memory** If you are using multiple Lookup Tables formulas in different cells on the same sheet and the same row you should use this option which stores the memory separately for each cell.
- **0 = Do not use Lookup memory**
- **Named Memory** If you give a text string instead of a number it will be used as a named memory. You can have many named memories in use at the same time (usually one for each Lookup Table). Each named memory works by row across all sheets within a workbook, so is the best choice when you have:
  - More than one memory lookup referring to different lookup tables within a single formula.
  - More than one memory lookup referring to different lookup tables in different cells on the same row, and these lookups are repeated on the same row in multiple sheets.

If this option is not zero, then MEMMATCH will first check to see if the index stored in memory that gave the answer the last time the MEMMATCH was calculated still gives the correct answer. If it does then MEMMATCH will return that answer without doing any more processing.

When using Lookup memory and there are multiple exact match answers available MEMMATCH will not necessarily return the answer from the same row as MATCH.

Vertical_Horizontal (Optional, Defaults to 1)

This parameter controls whether the lookup is done vertically or horizontally.

0 = Guess – if the number of rows >= the number of columns vertical, otherwise horizontal
1= Always Vertical (Default)
2= Always Horizontal

MEMMATCH Performance

If your data is **unsorted** and there is no Lookup Memory already available for the position of the MEMMATCH formula then MEMMATCH will do a linear search and store the row or column index found in the Lookup Memory for subsequent MEMMATCH calls.
In this case the second execution of the MEMMATCH formula will be much faster than the first.

If your data is **sorted** then MEMMATCH will do an **exact match Binary Search**, which is very fast, and the second execution of the MEMMATCH formula will also be very fast.

AVLOOKUP2, AVLOOKUPS2 & AVLOOKUPNTH Functions

Search for values in one or more columns of a table, and return values from the rows where a match is found.

Advanced Lookup functions returning:

- either the first value found (AVLOOKUP2)
- or all the values found (AVLOOKUPS2)
- or the Nth value found (AVLOOKUPNTH)

AVLOOKUP2, AVLOOKUPS2 and AVLOOKUPNTH are NOT case-sensitive.

The AVLOOKUP functions are multi-threaded, non-volatile, array functions.

AVLOOKUP Family Syntax

AVLOOKUP2(Lookup_Values, Lookup_Table, Answer_Columns, Sorted, Exact_Match, Lookup_Columns, MemType_Name)

The first 3 parameters are required; the last 4 parameters are optional.

AVLOOKUPS2 (Lookup_Values, Lookup_Table, Answer_Columns, Sorted, Exact_Match, Lookup_Columns)

The first 3 parameters are required; the last 3 parameters are optional.

AVLOOKUPNTH (Lookup_Values, Lookup_Table, Answer_Columns, Sorted, Exact_Match, Lookup_Columns, Position)

The first 3 parameters are required; the last 4 parameters are optional.

Lookup_Values (required)

The value(s) to be found in the Lookup_Columns.
Can be a single value or multiple values arranged in columns (lookup in multiple lookup columns) and rows (return multiple answer Rows). A single value can be a constant or a cell reference.
Multiple lookup values can be specified either as an array of constants or as a range referring to multiple cells. There should be the same number of columns of lookup values as there are columns in the Lookup_Columns.

When doing multi-column lookups (multiple columns of both lookup values and lookup columns) the AVLOOKUP functions will look for a row where ALL the lookup values are matched in the corresponding columns (Columns are treated as AND).
When doing multi-row lookups (multiple rows of lookup values) the AVLOOKUP functions will look for a row separately for each row of lookup values (Rows are treated as OR).
A single AVLOOKUP2 or AVLOOKUPNTH statement will return the same number of rows and columns of result values as there are rows in Lookup_Values and columns in Answer_Columns.

Lookup_Values can contain the wildcard characters ? and * for exact matches on unsorted text data. To find actual question marks or asterisks add a tilde (~) preceding the character.

Lookup_Table (required)

The rectangular range of cells or array or expression yielding an array to be used for the lookup table. The array/range must resolve to a single contiguous rectangular array.

Answer_Columns (required)

Specifies the column or columns in Lookup_Table that the AVLOOKUP functions will return values from for the row or rows that are found in the lookup operation.
Answer_Columns can be a constant, an array of constants, an expression or a reference.

- If Answer_Columns evaluates to a number it will be treated as relative column number(s) within Lookup_Table.
- If Answer_Columns evaluates to text then the text will be treated as column labels to be found in the first row of Lookup_Table.

If the column labels are not found the AVLOOKUP functions return #REF.

Sorted (optional, defaults to False)

Specifies whether the data in Lookup_Table is sorted on the first Lookup Column ascending, descending or is not sorted. Valid values for Sorted are:

- True, “Asc”, “Yes”, “True”, 1 Ascending
- “Des”, -1 Descending
- False, “No”, 0, any other text Not Sorted

If the Lookup_Table is sorted on the first Lookup_Column the lookup process will be significantly faster.

Exact_Match (optional, defaults to True)

Use this optional parameter when you want the AVLOOKUP functions to find a row in Table_range that exactly matches the Lookup_Value(s), even with sorted data, **and also to specify what to return if an exact match does not exist.**

An exact match will always be done with unsorted data.
A value of False means that an approximate match will be found with sorted data.
Use True to find an exact march with sorted data and return #N/A if not found.
An exact match on sorted data is much more efficient than an exact match on unsorted data.

If a value of anything other than False is specified it will indicate that an exact match is to be done even with sorted data, and the value given is the value to be returned if no exact match can be found.
If True is specified or the parameter is omitted the error value will be #N/A.

Lookup_Columns (optional, defaults to 1)

Use this optional parameter when you want to control which column(s) in Lookup_Table to use for the lookup.
The default value is 1: the first column in Lookup_Table

If the values are numeric they will be treated as column numbers, if alphabetic they will be treated as column labels to be found in the first row of Lookup_Table.
If there is more than one column given then there must be a matching set of values given in the columns of Lookup_Values.
Multiple columns can be given either as an array of constants or as a range reference.

MemType_Name (Optional, Defaults to 2, AVLOOKUP2 only)

Memory type for lookups can be 0, 1, 2 or 3 or a string that is used as the name for a memory.

- **3 = Global Memory for Rows or Columns** A single index is stored in global memory for each row (vertical lookups) or column (horizontal lookups).
  This global memory is super-efficient and easy to use when you have more than one lookup formula on the same row looking up different columns in a Table. This global memory works across all worksheets in all open workbooks.
- **2 = Book Sheet Row Memory** If you are using different Lookup Tables on different worksheets but still want to have more than one lookup formula on a row looking up different columns in the same table you can use this option which stores the row memory separately for each Workbook worksheet.

- **1 = Book Sheet Cell Memory** If you are using multiple Lookup Tables formulas in different cells on the same sheet and the same row you should use this option which stores the memory separately for each cell.
- **0 = Do not use Lookup memory**
- **Named Memory** If you give a text string instead of a number it will be used as a named memory. You can have many named memories in use at the same time (usually one for each Lookup Table). Each named memory works by row across all sheets within a workbook, so is the best choice when you have:
    - More than one memory lookup referring to different lookup tables within a single formula.
    - More than one memory lookup referring to different lookup tables in different cells on the same row, and these lookups are repeated on the same row in multiple sheets.

If this option is not zero, then AVLOOKUP2 will first check to see if the index stored in memory that gave the answer the last time the AVLOOKUP2 was calculated still gives the correct answer. If it does then AVLOOKUP2 will return that answer without doing any more processing.

When using Lookup memory and there are multiple exact match answers available AVLOOKUP2 will not necessarily return the answer from the same row as VLOOKUP.

AVLOOKUPS and AVLOOKUPNTH do not use Lookup Memory.

Position (Optional, Defaults to 0, AVLOOKUPNTH only)

Controls which result will be returned when there are multiple rows that match the Lookup_Values..

- N: where N is a positive integer. The Nth match found will be returned
- 0 : If sorted ascending the largest value that is less than or equal to Lookup_Value
  If not sorted then the first value found
  If sorted descending then the smallest value that is greater than or equal to lookup value
- -1: The first value found will always be returned
- -2: the Last Value found will always be returned
- -3: All matches found will be returned

Remarks

AVLOOKUP2 returns the first row that it finds which meets these criteria:

- Sorted Ascending – the largest value that is less than or equal to Lookup_Value
- Sorted Descending – the smallest value that is greater than or equal to Lookup_Value.
- Not Sorted – The first row containing a value equal to Lookup_Value, except when using the built-in memory function. In this case AVLOOKUP2 will return the same row as in the previous calculation provided it still matches the Lookup_Value.

AVLOOKUPS2 with one or more rows of Lookup_Values, and **AVLOOKUP2 or AVLOOKUPNTH** with multiple rows of Lookup Values can return a variable number of rows depending on how many rows meet the lookup criteria. So you should use them either in an array formula that returns multiple rows or embedded in functions like SUM(), AVERAGE() etc. that can handle arrays containing a variable number of results.

AVLOOKUP2 Examples

These examples can be found in the Examples subfolder of the FastExcel V5 SpeedTools install directory.

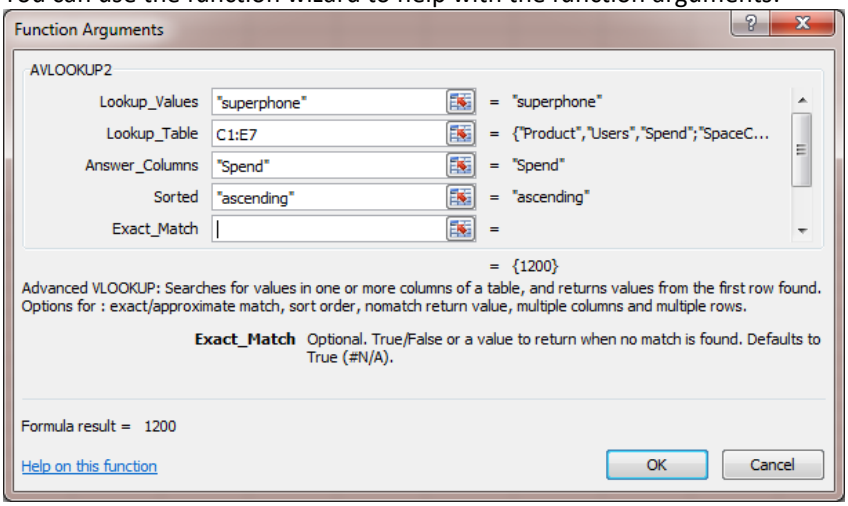
	A	B	C	D	E
1	**Channel**	**Sector**	**Product**	**Users**	**Spend**
2	Direct	Defense	SpaceCalls	7	1015
3	Direct	Mfg	SpaceCalls	2	205
4	Retail	Defense	SuperPhone	10	100
5	Retail	Mfg	SuperPhone	200	1000
6	Direct	Defense	SuperPhone	5	150
7	Direct	Mfg	SuperPhone	80	1200

Approx Lookup, ascending, exact result

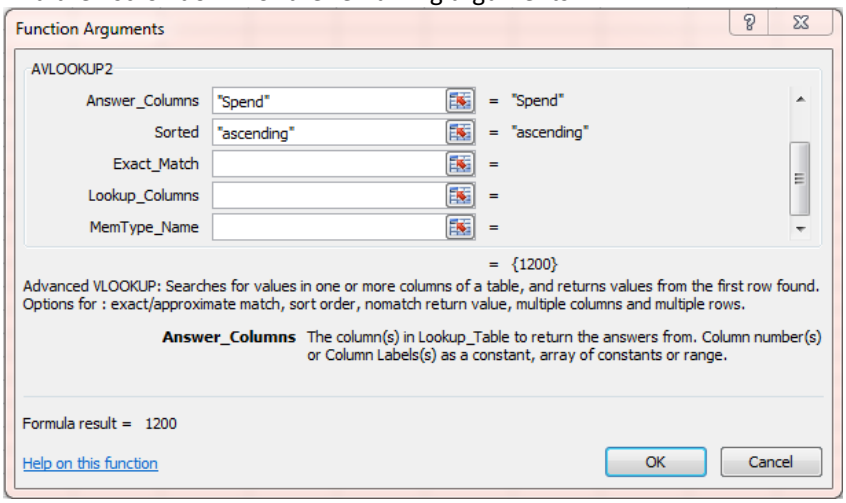
These examples look for “Superphone” in column C and return the corresponding value from column E.

Approx match returns the last result that matches.
VLOOKUP(“superphone”,C2:E7,3)=1200
AVLOOKUP2(“superphone”,c2:e7,3,true)=1200
AVLOOKUP2(“superphone”,c1:e7,”spend”,”asc”)=1200

You can use the function wizard to help with the function arguments:



And then scroll down for the remaining arguments:



Approx lookup, ascending, approx result

These examples look for “spacetime” in column C, but there is no “spacetime” so they find the row with the largest value less than “spacetime” and return the corresponding value from column E.

“spacetime” falls between “SpaceCalls” and “Superphone”, so row 3 is the largest value less than “spacetime”.
VLOOKUP(“spacetime”,c2:e7,3) =205

The default option for AVLOOKUP2 is to do an exact match even with sorted data, so #N/A is returned
AVLOOKUP2(“spacetime”,c1:e7,”spend”,1) = #N/A

You can make AVLOOKUP2 do an approximate match by giving FALSE as the 5th argument.
AVLOOKUP2(“spacetime”,c1:e7,”spend”,1,FALSE) =205

Exact match, not sorted, exact result

These examples look for “superphone” in column C and return the corresponding value from column C. Using AVLOOKUP2 you can either use the column number or the label of the column (“Spend”). Using the label gives you much better protection when someone inserts a column between Users and Spend.

Exact match returns the first result that exactly matches.
VLOOKUP(“superphone”,c2:e7,3,false) =100
AVLOOKUP2(“superphone”,c1:e7,”spend”,”No”) =100

Exact Match, sorted, not found

This example looks for an exact match to “spacetime” in column C, and cannot find one.

The default value returned if an exact match is not found is #N/A.
VLOOKUP(“spacetime”,c2:e7,3,false) = #N/A

The Sorted parameter is “No” so AVLOOKUP does an exact match and returns the default error #N/A.
AVLOOKUP2(“Spacetime”,C1:E7,”spend”,”No”) = #N/A

AVLOOKUP2 can also do an **exact match lookup on sorted data**, and it is much faster than an exact match on unsorted data, particularly with large ranges.
AVLOOKUP2(“Spacetime”,C1:E7,”spend”,”ASC”,True) = #N/A

Getting rid of the #N/A

If you want to return a value other than #N/A you can use IF tests with either COUNTIF or a double lookup.
IF(COUNTIF(C2:C7,”=spacetime”)=0,”None”, VLOOKUP(“Spacetime”,c2:e7,3,True)) = None

Using AVLOOKUP2 is simpler and faster because it allows you to set the error return value directly and can take advantage of sorted data.
AVLOOKUP2(“Spacetime”,c1:e7,”spend”,”asc”,”None”) = None

Lookup column not the first column

This example looks for “Superphone” in the “Product” column and returns the corresponding value from the “Spend” column.

To lookup a column that is not the first column you can use INDEX and MATCH.
INDEX(A2:E7,MATCH(“Superphone”,c2:c7,1),5) = 1200

With AVLOOKUP2 you can specify both the return column and the lookup column directly.
AVLOOKUP2("superphone",A1:E7,"spend",TRUE,FALSE,"product") = 1200

Two-dimensional lookup

This example looks for “Superphone” in column C and “Spend” in row 1, and returns the value of the intersection of the row and column found.

You can do a two-dimensional lookup using INDEX with two MATCH functions.
INDEX(A2:E7,MATCH(“Superphone”,c2:c7,1),MATCH(“Spend”,A1:E1,0)) =1200

AVLOOKUP2 handles two-dimensional lookups directly.
AVLOOKUP2(“Superphone”,A1:E7,”Spend”,True,False,”product”) = 1200

Error handling two-dimensional lookups using INDEX and MATCH is messy.
**IF(ISNA(INDEX(A2:E7,MATCH(“superphone”,C2:C7,1), MATCH(“spend”,A1:E1,0))),
”None”,INDEX(A2:E7,MATCH(“superphone”,C2:C7,1), MATCH(“spend”,A1:E1,0))) = 1200**

AVLOOKUP2 handles error handling for two-dimensional lookups directly and much more efficiently.
AVLOOKUP2(“Superphone”,A1:E7,”spend”,”Asc”,”None”,”product”) = 1200

Multicolumn and 2D lookup

This example looks for a row with “superphone” in the “product” column and “defense” in the “sector” column, and returns the value from the “Users” column. If a row that meets these conditions is not found, AVLOOKUP2 returns “None”.

Looking up multiple columns is simple and efficient with AVLOOKUP2.
AVLOOKUP2({“superphone”,”defense”},A1:E7,”Users”,”Asc”,”None”,{“product”,”sector”}) = 5
The two values to look up and the two columns to look up are specified as arrays using curly brackets. You should NOT enter this formula as an array formula.

Often it is better to reference ranges that contain the values rather than use arrays. This example assumes that G41:H41 contains “superphone” and “defense”, and I41:J41 contains “Product” and “Sector”.
AVLOOKUP2($G$41:$H$41,$A$1:$E$7,"users","ascending","None",$I$41:$J$41)=5

When using ranges containing the lookup values in this way the data needs to be in adjacent columns

Multicolumn lookup with columns that are not adjacent

When the columns containing the lookup values that you want to lookup are not next to one another you can use the COL.ARRAY function to create an array that makes the data adjacent.

So if you want to look up spend for a Sector and Channel the function would look like this:
AVLOOKUP2(COL.ARRAY(,C51,A51),A1:E7,"Spend","Asc","None",{"Sector","Channel"})
(note the , at the start of the COL.ARRAY function argument list: this is to use trhe default PAD argument)

Using Ranges, 6-cell array formula in D50:E52

	A	B	C	D	E
49	**Channel**	**Product**	**Sector**	**Spend**	**Users**
50	direct	superphone	defense	150	5
51	direct	superphone	mfg	1200	80
52	retail	spacecalls	mfg	none	none

This is an example of using AVLOOKUP2 as a multi-cell array formula. The formula is entered into D50:E52 as an array formula using Ctrl-Shift-Enter, and returns six results because there are three rows to lookup (50:52, each row has 3 columns to lookup) and two columns (D:E) to return from.

The example looks up “direct” “superphone” and “defense” in the columns labelled “Channel” “product” and “sector” and returns the values from the corresponding row in columns “Spend” and “Users” (150 and 5). It then looks up “direct” “superphone” and “mfg” and returns the values 1200 and 80, and finally looks up “retail” “spacecalls” and “mfg” to return the error values “none” and “none”.
{AVLOOKUP2(A50:C52,A1:E7,D49:E49,"Not","none",A49:C49)}
This multi-cell array formula is the equivalent of these six individual formulas:
AVLOOKUP2(A50:C50,A1:E7,D49,"Not","none",A49:C49)
AVLOOKUP2(A50:C50,A1:E7,E49,"Not","none",A49:C49)
AVLOOKUP2(A51:C51,A1:E7,D49,"Not","none",A49:C49)
AVLOOKUP2(A51:C51,A1:E7,E49,"Not","none",A49:C49)
AVLOOKUP2(A52:C52,A1:E7,D49,"Not","none",A49:C49)
AVLOOKUP2(A52:C52,A1:E7,E49,"Not","none",A49:C49)

Note that the sequence of the column names in the Lookup is not the same as the sequence of the column names in the data.

Returning multiple rows and columns using arrays and a 4-cell array formula

You can use arrays of constants or ranges in AVLOOKUP2. In this example the formula is Ctrl-shift-entered into A55:B56.

The first array of lookup values has two rows each containing two lookup values: (comma separates columns, semicolon separates rows).
The second array {spend, users} specifies the names of the two columns to return values from.
The third array {Product, sector} specifies the names of the two columns to look up.

{AVLOOKUP2({"Superphone","defense";"superphone","mfg"},A1:E7,{"spend","users"},"Asc","None",{"product","sector"})}

The results ofthe array formula are shown below:

	A	B
55	150	5
56	1200	80

Using SUM and wildcards with AVLOOKUPS2

This example uses AVLOOKUPS2 to return all the rows that match a simple lookup for Channel and Sector and a wildcard lookup for Product. The results are summed separately for Spend and for Users. If no matching rows are found then 0 will be returned.

D59= SUM(AVLOOKUPS2($A59:$C59,$A$1:$E$7,D$49,"Not",0,$A$49:$C$49))

Fill across to E59 and down to D60:E60

This is NOT an array formula and does not need to be entered using Ctrl-shift-Enter.

	A	B	C	D	E
49	**Channel**	**Product**	**Sector**	**Spend**	**Users**
59	direct	*P*	Mfg	1405	82
60	retail	*P*	Mfg	1000	200

The answer value of 1405 in D59 is the sum of the Spend column for all the rows that have Channel=Direct and Sector=Mfg and Product contains a P.

The result of 1405 in D59 comes from the rows returned to SUM as follows:

Row 3: Direct Mfg SpaceCalls = 205

Row 7: Direct Mfg SuperPhone = 1200

The other results are obtained in a similar way.

Case.AVLOOKUP2, Case.AVLOOKUPS2 & Case.AVLOOKUPNTH Functions

Search for values in one or more columns of a table, and return values from the rows where a match is found.

Advanced Lookup functions returning:

- either the first value found (Case.AVLOOKUP2)
- or all the values found (Case.AVLOOKUPS2)
- or the Nth value found (Case.AVLOOKUPNTH)

Case.AVLOOKUP2, Case.AVLOOKUPS2 and Case.AVLOOKUPNTH are case-sensitive versions of AVLOOKUP2, AVLOOKUPS2 and AVLOOKUPNTH respectively.

The Case.AVLOOKUP family of functions are multi-threaded, non-volatile array functions.

Case.AVLOOKUP Family Syntax

Case.AVLOOKUP2(Lookup_Values, Lookup_Table, Answer_Columns, Sorted, Exact_Match, Lookup_Columns, MemType_Name)

The first 3 parameters are required; the last 4 parameters are optional.

Case.AVLOOKUPS2(Lookup_Values, Lookup_Table, Answer_Columns, Sorted, Exact_Match, Lookup_Columns)

The first 3 parameters are required; the last 3 parameters are optional.

Case.AVLOOKUPNTH(Lookup_Values, Lookup_Table, Answer_Columns, Sorted, Exact_Match, Lookup_Columns, Position)

The first 3 parameters are required; the last 4 parameters are optional.

Lookup_Values (required)

The value(s) to be found in the Lookup_Columns.
Can be a single value or multiple values arranged in columns (lookup in multiple lookup columns) and rows (return multiple answer Rows).
A single value can be a constant or a cell reference.
Multiple lookup values can be specified either as an array of constants or as a range referring to multiple cells. There should be the same number of columns of lookup values as there are columns in the Lookup_Columns.

When doing multi-column lookups (multiple columns of both lookup values and lookup columns) the Case.AVLOOKUP functions will look for a row where ALL the lookup values are matched in the corresponding columns (Columns are treated as AND).

When doing multi-row lookups (multiple rows of lookup values) the Case.AVLOOKUP functions will look for a row separately for each row of lookup values (Rows are treated as OR).

A single Case.AVLOOKUP2 or Case.AVLOOKUPNTH statement will return the same number of rows and columns of result values as there are rows in Lookup_Values and columns in Answer_Columns.

Lookup_Values can contain the wildcard characters ? and * for exact matches on unsorted text data. To find actual question marks or asterisks add a tilde (~) preceding the character.

Lookup_Table (required)

The rectangular range of cells or array or expression yielding an array to be used for the lookup table. The array/range must resolve to a single contiguous rectangular array.

Answer_Columns (required)

Specifies the column or columns in Lookup_Table that the Case.AVLOOKUP functions will return values from for the row or rows that are found in the lookup operation.
Answer_Columns can be a constant, an array of constants, an expression or a reference.

- If Answer_Columns evaluates to a number it will be treated as relative column number(s) within Lookup_Table.
- If Answer_Columns evaluates to text then the text will be treated as column labels to be found in the first row of Lookup_Table.

If the column labels are not found the Case.AVLOOKUP functions return #REF.

Sorted (Optional, default False)

Specifies whether the data in Lookup_Table is sorted on the first Lookup Column ascending, descending or not sorted. Valid values for Sorted are:

- True, “Asc”, “Yes”, “True”, 1 Ascending
- “Des”, -1 Descending
- False, “No”, 0, any other text Not Sorted

If the Lookup_Table is sorted on the first Lookup_Column the lookup process will be significantly faster.

Exact_Match (optional, defaults to True)

Use this optional parameter when you want the Case.AVLOOKUP functions to find a row in Table_range that exactly matches the Lookup_Value(s), even with sorted data, **and also to specify what to return if an exact match does not exist.**

An exact match will always be done with unsorted data.

A value of False means that an approximate match will be found with sorted data.

Use True to find an exact march with sorted data and return #N/A if not found.
An exact match on sorted data is much more efficient than an exact match on unsorted data.
If a value of anything other than False is specified it will indicate that an exact match is to be done even with sorted data, and the value given specifies the value to be returned if no exact match can be found.
If True is specified or the parameter is omitted the error value will be #N/A.

Lookup_Columns (optional, defaults to 1)

Use this optional parameter when you want to control which column(s) in Lookup_Table to use for the lookup.
The default value is 1: the first column in Lookup_Table.
If the values are numeric they will be treated as column numbers, if alphabetic they will be treated as column labels to be found in the first row of Lookup_Table.
If there is more than one column given then there must be a matching set of values given in the columns of Lookup_Values.

Multiple columns can be given either as an array of constants or as a range reference.

MemType_Name (Optional, Defaults to 2, CASE.AVLOOKUP2 only)

Memory type for lookups can be 0, 1, 2 or 3 or a string that is used as the name for a memory.

- **3 = Global Memory for Rows or Columns** A single index is stored in global memory for each row (vertical lookups) or column (horizontal lookups).
  This global memory is super-efficient and easy to use when you have more than one lookup formula on the same row looking up different columns in a Table. This global memory works across all worksheets in all open workbooks.

- **2 = Book Sheet Row Memory** If you are using different Lookup Tables on different worksheets but still want to have more than one lookup formula on a row looking up different columns in the same table you can use this option which stores the row memory separately for each Workbook worksheet.
- **1 = Book Sheet Cell Memory** If you are using multiple Lookup Tables formulas in different cells on the same sheet and the same row you should use this option which stores the memory separately for each cell.
- **0 = Do not use Lookup memory**
- **Named Memory** If you give a text string instead of a number it will be used as a named memory. You can have many named memories in use at the same time (usually one for each Lookup Table). Each named memory works by row across all sheets within a workbook, so is the best choice when you have:
    - More than one memory lookup referring to different lookup tables within a single formula.
    - More than one memory lookup referring to different lookup tables in different cells on the same row, and these lookups are repeated on the same row in multiple sheets.

If this option is not zero, then Case.AVLOOKUP2 will first check to see if the index stored in memory that gave the answer the last time the Case.AVLOOKUP2 was calculated still gives the correct answer. If it does then Case.AVLOOKUP2 will return that answer without doing any more processing.

When using Lookup memory and there are multiple exact match answers available Case.AVLOOKUP2 will not necessarily return the answer from the same row as VLOOKUP.

Case.AVLOOKUPS and Case.AVLOOKUPNTH do not use Lookup Memory.

Position (Optional, Defaults to 0, Case.AVLOOKUPNTH only)

Controls which result will be returned when there are multiple rows that match the Lookup_Values..

- N: where N is a positive integer. The Nth match found will be returned
- 0 : If sorted ascending the largest value that is less than or equal to Lookup_Value
  If not sorted then the first value found
  If sorted descending then the smallest value that is greater than or equal to lookup value
- -1: The first value found will always be returned
- -2: the Last Value found will always be returned
- -3: All matches found will be returned

Remarks

Case.AVLOOKUP2 returns the first row that it finds which meets these criteria:

- Sorted Ascending – the largest value that is less than or equal to Lookup_Value
- Sorted Descending – the smallest value that is greater than or equal to Lookup_Value.
- Not Sorted – The first row containing a value equal to Lookup_Value, except when using the built-in memory function. In this case Case.AVLOOKUP2 will return the same row as in the previous calculation provided it still matches the Lookup_Value.

Case.AVLOOKUPS2 with one or more rows of Lookup_Values, and **Case.AVLOOKUP2 or Case.AVLOOKUPNTH** with multiple rows of Lookup Values can return a variable number of rows depending on how many rows meet the lookup criteria. So you should use them either in a dynamic array formula that returns multiple rows or embedded in functions like SUM(), AVERAGE() etc. that can handle arrays containing a variable number of results.

AMATCH2, AMATCHES2 & AMATCHNTH functions

Search for values in one or more columns of a table, and return the relative position of the row(s) where a match is found.

Advanced MATCH functions returning:

- either the relative position of the first value found (AVMATCH2)
- or the relative position of all the values found (AVMATCHES2)
- or the relative position of the Nth value found (AMATCHNTH)

These functions are similar to the AVLOOKUP2, AVLOOKUPS2 and AVLOOKUPNTH functions except that they return relative row numbers rather than values.

These functions are NOT case-sensitive.

The AMATCH functions are multi-threaded, non-volatile array functions.

AMATCH Family Syntax

AMATCH2 (Lookup_Values, Lookup_Table, Sorted, Exact_Match, Lookup_Columns, MemType_Name)

The first 2 parameters are required; the last 4 parameters are optional.

AMATCHES2 (Lookup_Values, Lookup_Table, Sorted, Exact_Match, Lookup_Columns)

The first 2 parameters are required; the last 3 parameters are optional.

AMATCHNTH (Lookup_Values, Lookup_Table, Sorted, Exact_Match, Lookup_Columns, Position)

The first 2 parameters are required; the last 4 parameters are optional.

AMATCH2, AMATCHES2 and AMATCHNTH return row number(s) within the range specified by Lookup_Table.

Lookup_Values (required)

Specifies the value(s) to be found in the Lookup_Columns. Can be a single value or multiple values arranged in columns (multiple lookup columns) and rows (multiple Lookup Rows).
A single value can be a constant or a range reference.
Multiple lookup values can be specified either as an array of constants or as a range referring to multiple cells. There should be the same number of columns of lookup values as there are columns in the Lookup_Columns.

When doing multi-column lookups (multiple columns of both lookup values and lookup columns) the AMATCH functions will look for a row where ALL the lookup values are matched in the corresponding columns (Columns are treated as AND).

When doing multi-row lookups (multiple rows of lookup values) the AMATCH functions will look for a row separately for each row of lookup values (Rows are treated as OR).

A single AMATCH2 or AMATCHNTH statement will return the same number of row numbers as there are rows in Lookup_Values.

Lookup_Values can contain the wildcard characters ? and * for exact matches on unsorted text data. To find actual question marks or asterisks add a tilde (~) preceding the character.

Lookup_Table (required)

The rectangular range of cells or array or expression yielding an array to be used for the lookup table. The array/range must resolve to a single contiguous rectangular array.

Sorted (Optional, defaults to False)

Specifies whether the data in Lookup_Table is sorted on the first Lookup Column ascending, descending or not sorted. Valid values for Sorted are:

- True, “Asc”, “Yes”, “True”, 1 Ascending
- “Des”, -1 Descending
- False, “No”, 0, any other text Not Sorted

If the Lookup_Table is sorted on the first Lookup_Column the lookup process will be significantly faster.

Exact_Error (optional, defaults to True)

Use this optional parameter when you want the AMATCH functions to find a row in Lookup_Table that exactly matches the Lookup_Value(s), even with sorted data, and to specify what to return if an exact match does not exist.

An exact match will always be done with unsorted data.

A value of False means that an approximate match will be found with sorted data.

Use True to find an exact march with sorted data and return #N/A if not found.

An exact match on sorted data is much more efficient than an exact match on unsorted data.

If a value of anything other than False is specified it will indicate that an exact match is to be done even with sorted data, and the value given specifies the value to be returned if no exact match can be found. If True is specified or the parameter is omitted the error value will be #N/A.

Lookup_Columns (optional, defaults to 1)

Use this optional parameter when you want to control which column(s) in Lookup_Table to use for the lookup.
The default value is 1: the first column in Lookup_Table

If the values are numeric, they will be treated as column numbers, if alphabetic they will be treated as column labels to be found in the first row of Lookup_Table.

If there is more than one column given then there must be a matching set of values given in the columns of Lookup_Values.

Multiple columns can be given either as an array of constants or as a range reference.

MemType_Name (Optional, Defaults to 2, AMATCH2 only)

Memory type for lookups can be 0, 1, 2 or 3 or a string that is used as the name for a memory.

- **3 = Global Memory for Rows or Columns** A single index is stored in global memory for each row (vertical lookups) or column (horizontal lookups).
  This global memory is super-efficient and easy to use when you have more than one lookup formula on the same row looking up different columns in a Table. This global memory works across all worksheets in all open workbooks.
- **2 = Book Sheet Row Memory** If you are using different Lookup Tables on different worksheets but still want to have more than one lookup formula on a row looking up different columns in the same table you can use this option which stores the row memory separately for each Workbook worksheet.
- **1 = Book Sheet Cell Memory** If you are using multiple Lookup Tables formulas in different cells on the same sheet and the same row you should use this option which stores the memory separately for each cell.
- **0 = Do not use Lookup memory**
- **Named Memory** If you give a text string instead of a number it will be used as a named memory. You can have many named memories in use at the same time (usually one for each Lookup Table). Each named memory works by row across all sheets within a workbook, so is the best choice when you have:
  - More than one memory lookup referring to different lookup tables within a single formula.
  - More than one memory lookup referring to different lookup tables in different cells on the same row, and these lookups are repeated on the same row in multiple sheets.

If this option is not zero, then AMATCH2 will first check to see if the index stored in memory that gave the answer the last time the AMATCH2 was calculated still gives the correct answer. If it does then AMATCH2 will return that answer without doing any more processing.

When using Lookup memory and there are multiple exact match answers available AMATCH2 will not necessarily return the answer from the same row as MATCH.

AMATCHES2 and AMATCHNTH do not use lookup memory.

Position (Optional, Defaults to 0, AMATCHNTH only)

Controls which result will be returned from multiple matches.

- N: where N is a positive integer. The Nth match found will be returned
- 0 : If sorted ascending then the largest value that is less than or equal to Lookup_Value
  If not sorted then the first value found
  If sorted descending then the smallest value that is greater than or equal to lookup value
- -1: The first value found will always be returned
- -2: the Last Value found will always be returned
- -3: All matches found will be returned

Remarks

AMATCH2 returns the first row number that it finds which meets these criteria:

- Sorted Ascending – the largest value that is less than or equal to Lookup_Value
- Sorted Descending – the smallest value that is greater than or equal to Lookup_Value.
- Not Sorted – The first row containing a value equal to Lookup_Value, except when using the built-in memory function. In this case AMATCH2 will return the same row as in the previous calculation provided it still matches the Lookup_Value.

AMATCHES2 with one or more rows of Lookup_Values, and **AMATCH2 or AMATCHNTH** with multiple rows of Lookup Values can return a variable number of rows depending on how many rows meet the lookup criteria. So you should use them either in a dynamic array formula that returns multiple rows or embedded in functions like SUM(), AVERAGE() etc. that can handle arrays containing a variable number of results.

Case.AMATCH2, Case.AMATCHES2 & Case.AMATCHNTH functions

Search for values in one or more columns of a table, and return the relative position of the row(s) where a match is found.

Advanced MATCH functions returning:

- either the relative position of the first value found (Case.AMATCH2)
- or the relative position of all the values found (Case.AMATCHES2)
- or the relative position of the Nth value found (Case.AMATCHNTH)

These functions are similar to the Case.AVLOOKUP2, Case.AVLOOKUPS2 and Case.AVLOOKUPNTH functions except that they return relative row numbers rather than values.

These functions are case-sensitive versions of AMATCH2, AMATCHES2 and AMATCHNTH

The Case.AMATCH functions are multi-threaded, non-volatile array functions.

Case.AMATCH Family Syntax

Case.AMATCH2(Lookup_Values, Lookup_Table, Sorted, Exact_Match, Lookup_Columns, MemType_Name)

The first 2 parameters are required; the last 4 parameters are optional.

Case.AMATCHES2(Lookup_Values, Lookup_Table, Sorted, Exact_Match, Lookup_Columns)

The first 2 parameters are required; the last 3 parameters are optional.

Case.AMATCHNTH(Lookup_Values, Lookup_Table, Sorted, Exact_Match, Lookup_Columns, Position)

The first 2 parameters are required; the last 4 parameters are optional.

Case.AMATCH2, Case.AMATCHES2 and Case.AMATCHNTH return the relative position of row(s) within the range specified by Lookup_Table.

Lookup_Values (required)

Specifies the value(s) to be found in the Lookup_Columns. Can be a single value or multiple values arranged in columns (multiple lookup columns) and rows (multiple Lookup Rows).
A single value can be a constant or a range reference.
Multiple lookup values can be specified either as an array of constants or as a range referring to multiple cells. There should be the same number of columns of lookup values as there are columns in the Lookup_Columns.

When doing multi-column lookups (multiple columns of both lookup values and lookup columns) the Case.AMATCH functions will look for a row where ALL the lookup values are matched in the corresponding columns (Columns are treated as AND).

When doing multi-row lookups (multiple rows of lookup values) the Case.AMATCH functions will look for a row separately for each row of lookup values (Rows are treated as OR).

A single Case.AMATCH2 or CaseAMATCHNTH statement will return the same number of row numbers as there are rows in Lookup_Values.

Lookup_Values can contain the wildcard characters ? and * for exact matches on unsorted text data. To find actual question marks or asterisks add a tilde (~) preceding the character.

Lookup_Table (required)

The rectangular range of cells or array or expression yielding an array to be used for the lookup table. The array/range must resolve to a single contiguous rectangular array.

Sorted (Optional, defaults to False)

Specifies whether the data in Lookup_Table is sorted on the first Lookup Column ascending, descending or not sorted. Valid values for Sorted are:

- True, “Asc”, “Yes”, “True”, 1 Ascending
- “Des”, -1 Descending
- False, “No”, 0, any other text Not Sorted

If the Lookup_Table is sorted on the first Lookup_Column the lookup process will be significantly faster.

Exact_Error (optional, defaults to True)

Use this optional parameter when you want the Case.AMATCH functions to find a row in Lookup_Table that exactly matches the Lookup_Value(s), even with sorted data, and to specify what to return if an exact match does not exist.

An exact match will always be done with unsorted data.

A value of False means that an approximate match will be found with sorted data.

Use True to find an exact march with sorted data and return #N/A if not found.

An exact match on sorted data is much more efficient than an exact match on unsorted data.

If a value of anything other than False is specified it will indicate that an exact match is to be done even with sorted data, and the value given specifies the value to be returned if no exact match can be found. If True is specified or the parameter is omitted the error value will be #N/A.

Lookup_Columns (optional, defaults to 1)

Use this optional parameter when you want to control which column(s) in Lookup_Table to use for the lookup.
The default value is 1: the first column in Lookup_Table

If the values are numeric, they will be treated as column numbers, if alphabetic they will be treated as column labels to be found in the first row of Lookup_Table.

If there is more than one column given then there must be a matching set of values given in the columns of Lookup_Values.

Multiple columns can be given either as an array of constants or as a range reference.

MemType_Name (Optional, Defaults to 2, CASE.AMATCH2 only)

Memory type for lookups can be 0, 1, 2 or 3 or a string that is used as the name for a memory.

- **3 = Global Memory for Rows or Columns** A single index is stored in global memory for each row (vertical lookups) or column (horizontal lookups).
  This global memory is super-efficient and easy to use when you have more than one lookup formula on the same row looking up different columns in a Table. This global memory works across all worksheets in all open workbooks.
- **2 = Book Sheet Row Memory** If you are using different Lookup Tables on different worksheets but still want to have more than one lookup formula on a row looking up different columns in the same table you can use this option which stores the row memory separately for each Workbook worksheet.
- **1 = Book Sheet Cell Memory** If you are using multiple Lookup Tables formulas in different cells on the same sheet and the same row you should use this option which stores the memory separately for each cell.
- **0 = Do not use Lookup memory**
- **Named Memory** If you give a text string instead of a number it will be used as a named memory. You can have many named memories in use at the same time (usually one for each Lookup Table). Each named memory works by row across all sheets within a workbook, so is the best choice when you have:
  - More than one memory lookup referring to different lookup tables within a single formula.
  - More than one memory lookup referring to different lookup tables in different cells on the same row, and these lookups are repeated on the same row in multiple sheets.

If this option is not zero, then Case.AMATCH2 will first check to see if the index stored in memory that gave the answer the last time the Case.AMATCH2 was calculated still gives the correct answer. If it does then Case.AMATCH2 will return that answer without doing any more processing.

When using Lookup memory and there are multiple exact match answers available Case.AMATCH2 will not necessarily return the answer from the same row as MATCH.

Case.AMATCHES2 and Case.AMATCHNTH do not use lookup memory.

Position (Optional, Defaults to 0, Case.AMATCHNTH only)

Controls which result will be returned from multiple matches.

- N: where N is a positive integer. The Nth match found will be returned
- 0 : If sorted ascending the largest value that is less than or equal to Lookup_Value
  If not sorted then the first value found
  If sorted descending then the smallest value that is greater than or equal to lookup value
- -1: The first value found will always be returned
- -2: the Last Value found will always be returned
- -3: All matches found will be returned

Remarks

Case.AMATCH2 returns the first row number that it finds which meets these criteria:

- Sorted Ascending – the largest value that is less than or equal to Lookup_Value
- Sorted Descending – the smallest value that is greater than or equal to Lookup_Value.
- Not Sorted – The first row containing a value equal to Lookup_Value, except when using the built-in memory function. In this case Case.AMATCH2 will return the same row as in the previous calculation provided it still matches the Lookup_Value.

Case.AMATCHES2 with one or more rows of Lookup_Values, and **Case.AMATCH2 or Case.AMATCHNTH** with multiple rows of Lookup Values can return a variable number of rows depending on how many rows meet the lookup criteria. So you should use them either in a dynamic array formula that returns multiple rows or embedded in functions like SUM(), AVERAGE() etc. that can handle arrays containing a variable number of results.

Rgx.AVLOOKUP2, Rgx.AVLOOKUPS2 & Rgx.AVLOOKUPNTH Functions

Uses Regular Expressions to search for values in one or more columns of a table, and return values from the rows where a match is found.

Advanced Lookup functions returning:

- either the first value found (Rgx.AVLOOKUP2)
- or all the values found (Rgx.AVLOOKUPS2)
- or the Nth value found (Rgx.AVLOOKUPNTH)

Rgx.AVLOOKUP2, Rgx.AVLOOKUPS2 and Rgx.AVLOOKUPNTH are NOT case-sensitive.
These functions are the Regular Expression versions of AVLOOKUP2, AVLOOKUPS2 and AVLOOKUPNTH

The Rgx.AVLOOKUP family of functions are multi-threaded, non-volatile array functions.

Rgx.AVLOOKUP Family Syntax

Rgx.AVLOOKUP2(RegExp, Lookup_Table, Answer_Columns, Not_Found, Lookup_Columns, MemType_Name)

The first 3 parameters are required; the last 3 parameters are optional.

Rgx.AVLOOKUPS2(RegExp, Lookup_Table, Answer_Columns, Not_Found, Lookup_Columns)

The first 3 parameters are required; the last 2 parameters are optional.

Rgx.AVLOOKUPNTH(RegExp, Lookup_Table, Answer_Columns, Not_Found, Lookup_Columns, Position)

The first 3 parameters are required; the last 3 parameters are optional.

RegExp (required)

Specifies the Regular Expression Pattern(s) to be matched against the value in the Lookup_Column(s).

Can be a single Regular Expression or multiple Regular Expressions arranged in columns (multiple lookup columns) and rows (multiple Lookup Rows).
A single Regular Expression can be a constant or a cell reference.
Multiple lookup Regular Expressions can be specified either as an array of constants or as a range referring to multiple cells. There should be the same number of columns of Regular Expression values as there are columns in the Lookup_Columns.

When doing multi-column lookups (multiple columns of both lookup values and lookup columns) the Rgx.AVLOOKUP functions will look for a row where ALL the lookup Regular Expression patterns are matched in the corresponding columns (Columns are treated as AND).

When doing multi-row lookups (multiple rows of lookup values) the Rgx.AVLOOKUP functions will look for a row separately for each row of lookup values (Rows are treated as OR).

A single Rgx.AVLOOKUP2 or Rgx.AVLOOKUPNTH statement will return the same number of rows and columns of result values as there are rows and columns in RegExp.

Lookup_Table (required)

The rectangular range of cells or array or expression yielding an array to be used for the lookup table. The array/range must resolve to a single contiguous rectangular array.

Answer_Columns (required)

Specifies the column or columns in Lookup_Table that Rgx.AVLOOKUP2 will return values from for the row or rows that are found in the lookup operation.
Can be a constant, an array of constants or an expression that returns a range.

If Answer_Columns is an expression that returns a number it will be treated as column number(s) within Lookup_Table.
If Answer_Columns returns text this will be treated as column labels to be found in the first row of Lookup_Table. If the column labels are not found Rgx.AVLOOKUP2 returns #REF.

Not_Found (optional, defaults to #N/A)

Use this optional parameter to specify what to return if an exact match does not exist.
If the parameter is omitted the error value will be #N/A.

Lookup_Columns (optional, defaults to 1)

Use this optional parameter when you want to control which column(s) in Lookup_Table to use for the lookup.
The default value is 1: the first column in Lookup_Table

If the values are numeric they will be treated as column numbers, if alphabetic they will be treated as column labels to be found in the first row of Lookup_Table.

If there is more than one column given then there must be a matching set of values given in the columns of RegExp.

Multiple columns can be given either as an array of constants or as a range reference.

MemType_Name (Optional, Defaults to 2, Rgx.AVLOOKUP2 only)

Memory type for lookups can be 0, 1, 2 or 3 or a string that is used as the name for a memory.

- **3 = Global Memory for Rows or Columns** A single index is stored in global memory for each row (vertical lookups) or column (horizontal lookups).
  This global memory is super-efficient and easy to use when you have more than one lookup formula on the same row looking up different columns in a Table. This global memory works across all worksheets in all open workbooks.
- **2 = Book Sheet Row Memory** If you are using different Lookup Tables on different worksheets but still want to have more than one lookup formula on a row looking up different columns in the same table you can use this option which stores the row memory separately for each Workbook worksheet.
- **1 = Book Sheet Cell Memory** If you are using multiple Lookup Tables formulas in different cells on the same sheet and the same row you should use this option which stores the memory separately for each cell.
- **0 = Do not use Lookup memory**
- **Named Memory** If you give a text string instead of a number it will be used as a named memory. You can have many named memories in use at the same time (usually one for each Lookup Table). Each named memory works by row across all sheets within a workbook, so is the best choice when you have:
  - More than one memory lookup referring to different lookup tables within a single formula.
  - More than one memory lookup referring to different lookup tables in different cells on the same row, and these lookups are repeated on the same row in multiple sheets.

If this option is not zero, then Rgx.AVLOOKUPS2 will first check to see if the index stored in memory that gave the answer the last time the Rgx.AVLOOKUPS2 was calculated still gives the correct answer. If it does then Rgx.AVLOOKUPS2 will return that answer without doing any more processing.

When using Lookup memory and there are multiple exact match answers available Rgx.AVLOOKUPS2 will not necessarily return the answer from the same row as MATCH.

Rgx.AVLOOKUPS2 and Rgx.AVLOOKUPNTH do not use lookup memory.

Position (Optional, Defaults to 0, Rgx.AVLOOKUPNTH only)

Controls which result will be returned from multiple matches.

- N: where N is a positive integer. The Nth match found will be returned
- 0 : The first value found will always be returned
- -1: The first value found will always be returned
- -2: the Last Value found will always be returned
- -3: All matches found will be returned

Remarks

Rgx.AVLOOKUPS2 with one or more rows of Lookup_Values, and **Rgx.AVLOOKUP2 or Rgx.AVLOOKUPNTH** with multiple rows of Lookup Values can return a variable number of rows depending on how many rows meet the lookup criteria. So you should use them either in a dynamic array formula that returns multiple rows or embedded in functions like SUM(), AVERAGE() etc. that can handle arrays containing a variable number of results.

Rgx.Case.AVLOOKUP2, Rgx.Case.AVLOOKUPS2 & Rgx.Case.AVLOOKUPNTH Functions

Search for values in one or more columns of a table, and return values from the rows where a match is found.

Advanced Lookup functions returning:

- either the first value found (Rgx.Case.AVLOOKUP2)
- or all the values found (Rgx.Case.AVLOOKUPS2)
- or the Nth value found (Rgx.Case.AVLOOKUPNTH)

Rgx.Case.AVLOOKUP2, Rgx.Case.AVLOOKUPS2 and Rgx.Case.AVLOOKUPNTH are case-sensitive versions of Rgx.AVLOOKUP2, Rgx.AVLOOKUPS2 and Rgx.AVLOOKUPNTH respectively.

The Rgx.Case.AVLOOKUP family of functions are multi-threaded, non-volatile array functions.

Rgx.Case.AVLOOKUP Family Syntax

Rgx.Case.AVLOOKUP2 (RegExp, Lookup_Table, Answer_Columns, Not_Found, Lookup_Columns,MemType_Name)

The first 4 parameters are required; the last 3 parameters are optional.

Rgx.Case.AVLOOKUPS2 (RegExp, Lookup_Table, Answer_Columns, Not_Found, Lookup_Columns)

The first 4 parameters are required; the last 2 parameters are optional.

Rgx.Case.AVLOOKUPNTH (RegExp, Lookup_Table, Answer_Columns, Not_Found, Lookup_Columns, Position)

The first 4 parameters are required; the last 3 parameters are optional.

RegExp (required)

Specifies the Regular Expression Pattern(s) to be matched against the value in the Lookup_Column(s).

Can be a single Regular Expression or multiple Regular Expressions arranged in columns (multiple lookup columns) and rows (multiple Lookup Rows).
A single Regular Expression can be a constant or a cell reference.
Multiple lookup Regular Expressions can be specified either as an array of constants or as a range referring to multiple cells. There should be the same number of columns of Regular Expression values as there are columns in the Lookup_Columns.

When doing multi-column lookups (multiple columns of both lookup values and lookup columns) the Rgx.Case.AVLOOKUP functions will look for a row where ALL the lookup Regular Expression patterns are matched in the corresponding columns (Columns are treated as AND).

When doing multi-row lookups (multiple rows of lookup values) the Rgx.Case.AVLOOKUP functions will look for a row separately for each row of lookup values (Rows are treated as OR).

A single Rgx.Case.AVLOOKUP2 or Rgx.Case.AVLOOKUPNTH statement will return the same number of rows and columns of result values as there are rows and columns in RegExp.

Lookup_Table (required)

The rectangular range of cells or array or expression yielding an array to be used for the lookup table. The array/range must resolve to a single contiguous rectangular array.

Answer_Columns (required)

Specifies the column or columns in Lookup_Table that Rgx.Case.AVLOOKUP2 will return values from for the row or rows that are found in the lookup operation.
Can be a constant, an array of constants or an expression that returns a range.
If Answer_Columns is an expression that returns a number it will be treated as column number(s) within Lookup_Table.
If Answer_Columns returns text this will be treated as column labels to be found in the first row of Lookup_Table. If the column labels are not found Rgx.Case.AVLOOKUP2 returns #REF.

Not_Found (optional, defaults to #N/A)

Use this optional parameter to specify what to return if an exact match does not exist.
If the parameter is omitted the error value will be #N/A.

Lookup_Columns (optional, defaults to 1)

Use this optional parameter when you want to control which column(s) in Lookup_Table to use for the lookup.
The default value is 1: the first column in Lookup_Table

If the values are numeric they will be treated as column numbers, if alphabetic they will be treated as column labels to be found in the first row of Lookup_Table.

If there is more than one column given then there must be a matching set of values given in the columns of RegExp.

Multiple columns can be given either as an array of constants or as a range reference.

MemType_Name (Optional, Defaults to 2, Rgx.Case.AVLOOKUP2 only)

Memory type for lookups can be 0, 1, 2 or 3 or a string that is used as the name for a memory.

- **3 = Global Memory for Rows or Columns** A single index is stored in global memory for each row (vertical lookups) or column (horizontal lookups).
  This global memory is super-efficient and easy to use when you have more than one lookup formula on the same row looking up different columns in a Table. This global memory works across all worksheets in all open workbooks.
- **2 = Book Sheet Row Memory** If you are using different Lookup Tables on different worksheets but still want to have more than one lookup formula on a row looking up different columns in the same table you can use this option which stores the row memory separately for each Workbook worksheet.
- **1 = Book Sheet Cell Memory** If you are using multiple Lookup Tables formulas in different cells on the same sheet and the same row you should use this option which stores the memory separately for each cell.
- **0 = Do not use Lookup memory**
- **Named Memory** If you give a text string instead of a number it will be used as a named memory. You can have many named memories in use at the same time (usually one for each Lookup Table). Each named memory works by row across all sheets within a workbook, so is the best choice when you have:
  - More than one memory lookup referring to different lookup tables within a single formula.
  - More than one memory lookup referring to different lookup tables in different cells on the same row, and these lookups are repeated on the same row in multiple sheets.

If this option is not zero, then Rgx.Case.AVLOOKUPS2 will first check to see if the index stored in memory that gave the answer the last time the Rgx.Case.AVLOOKUPS2 was calculated still gives the correct

answer. If it does then Rgx.Case.AVLOOKUPS2 will return that answer without doing any more processing.

When using Lookup memory and there are multiple exact match answers available Rgx.Case.AVLOOKUPS2 will not necessarily return the answer from the same row as MATCH.

Rgx.Case.AVLOOKUPS2 and Rgx.Case.AVLOOKUPNTH do not use lookup memory.

Position (Optional, Defaults to 0, Rgx.Case.AVLOOKUPNTH only)

Controls which result will be returned from multiple matches.

- N: where N is a positive integer. The Nth match found will be returned
- 0 : The first value found will always be returned
- -1: The first value found will always be returned
- -2: the Last Value found will always be returned
- -3: All matches found will be returned

Remarks

Rgx.Case.AVLOOKUPS2 with one or more rows of Lookup_Values, and **Rgx.Case.AVLOOKUP2 or Rgx.Case.AVLOOKUPNTH** with multiple rows of Lookup Values can return a variable number of rows depending on how many rows meet the lookup criteria. So you should use them either in a dynamic array formula that returns multiple rows or embedded in functions like SUM(), AVERAGE() etc. that can handle arrays containing a variable number of results.

Rgx.AMATCH2, Rgx.AMATCHES2 & Rgx.AMATCHNTH functions

Search for values in one or more columns of a table, and return the relative position of the row(s) where a match is found.

Advanced MATCH functions returning:

- either the relative position of the first value found (Rgx.AMATCH2)
- or the relative position of all the values found (Rgx.AMATCHES2)
- or the relative position of the Nth value found (Rgx.AMATCHNTH)

These functions are similar to the Rgx.AVLOOKUP2, Rgx.AVLOOKUPS2 and Rgx.AVLOOKUPNTH functions except that they return row numbers rather than values.

These functions are NOT case-sensitive.

The Rgx.AMATCH family of functions are multi-threaded, non-volatile array functions.

Rgx.AMATCH Family Syntax

Rgx.AMATCH2(RegExp, Lookup_Table, Not_Found, Lookup_Columns, MemType_Name)

The first 3 parameters are required; the last 3 parameters are optional.

Rgx.AMATCHES2(RegExp, Lookup_Table, Lookup_Columns)

The first 2 parameters are required, the last parameter is optional.

Rgx.AMATCHNTH(RegExp, Lookup_Table, Not_Found, Lookup_Columns, Position)

The first 2 parameters are required; the last 3 parameters are optional.

Rgx.AMATCH2, Rgx.AMATCHES2 and Rgx.AMATCHNTH return row number(s) within the range specified by Lookup_Table.

RegExp (required)

Specifies the Regular Expression Pattern(s) to be matched against the value in the Lookup_Column(s).

Can be a single Regular Expression or multiple Regular Expressiones arranged in columns (multiple lookup columns) and rows (multiple Lookup Rows).
A single Regular Expression can be a constant or a cell reference.
Multiple lookup Regular Expressions can be specified either as an array of constants or as a range referring to multiple cells. There should be the same number of columns of Regular Expression values as there are columns in the Lookup_Columns.

When doing multi-column lookups (multiple columns of both lookup values and lookup columns) the Rgx.AMATCH functions will look for a row where ALL the lookup Regular Expression patterns are matched in the corresponding columns (Columns are treated as AND).

When doing multi-row lookups (multiple rows of lookup values) the Rgx.AMATCH functions will look for a row separately for each row of lookup values (Rows are treated as OR).

A single Rgx.AMATCH2 or Rgx.AMATCHNTH statement will return the same number of rows and columns of result values as there are rows and columns in p.

Lookup_Table (required)

The rectangular range of cells or array or expression yielding an array to be used for the lookup table. The array/range must resolve to a single contiguous rectangular array.

Not_Found (optional, defaults to False, Rgx.AMATCH2 and Rgx.AMATCHNTH only)

Use this optional parameter to specify what to return if an exact match does not exist.

If the parameter is omitted the error value will be #N/A.

Lookup_Columns (optional, defaults to 1)

Use this optional parameter when you want to control which column(s) in Lookup_Table to use for the lookup.
The default value is 1: the first column in Lookup_Table

If the values are numeric they will be treated as column numbers, if alphabetic they will be treated as column labels to be found in the first row of Lookup_Table.

If there is more than one column given then there must be a matching set of values given in the columns of RegExp.

Multiple columns can be given either as an array of constants or as a range reference.

MemType_Name (Optional, Defaults to 2, Rgx.AMATCH2 only)

Memory type for lookups can be 0, 1, 2 or 3 or a string that is used as the name for a memory.

- **3 = Global Memory for Rows or Columns** A single index is stored in global memory for each row (vertical lookups) or column (horizontal lookups).
  This global memory is super-efficient and easy to use when you have more than one lookup formula on the same row looking up different columns in a Table. This global memory works across all worksheets in all open workbooks.
- **2 = Book Sheet Row Memory** If you are using different Lookup Tables on different worksheets but still want to have more than one lookup formula on a row looking up different columns in the same table you can use this option which stores the row memory separately for each Workbook worksheet.
- **1 = Book Sheet Cell Memory** If you are using multiple Lookup Tables formulas in different cells on the same sheet and the same row you should use this option which stores the memory separately for each cell.
- **0 = Do not use Lookup memory**
- **Named Memory** If you give a text string instead of a number it will be used as a named memory. You can have many named memories in use at the same time (usually one for each Lookup Table). Each named memory works by row across all sheets within a workbook, so is the best choice when you have:
  - More than one memory lookup referring to different lookup tables within a single formula.
  - More than one memory lookup referring to different lookup tables in different cells on the same row, and these lookups are repeated on the same row in multiple sheets.

If this option is not zero, then Rgx.AMATCH2 will first check to see if the index stored in memory that gave the answer the last time the Rgx.AMATCH2 was calculated still gives the correct answer. If it does then Rgx.AMATCH2 will return that answer without doing any more processing.

When using Lookup memory and there are multiple exact match answers available Rgx.AMATCH2 will not necessarily return the answer from the same row as MATCH.

Rgx.AMATCHES2 and Rgx.AMATCHNTH do not use lookup memory.

Position (Optional, Defaults to 0, Rgx.AMATCHNTH only)

Controls which result will be returned from multiple matches.

- N: where N is a positive integer. The Nth match found will be returned
- 0 : The first value found will always be returned
- -1: The first value found will always be returned
- -2: the Last Value found will always be returned
- -3: All matches found will be returned

Remarks

Rgx.AMATCHES2 with one or more rows of Lookup_Values, and **Rgx.AMATCH2 or Rgx.AMATCHNTH** with multiple rows of Lookup Values can return a variable number of rows depending on how many rows meet the lookup criteria. So you should use them either in adynmic array formula that returns multiple rows or embedded in functions like SUM(), AVERAGE() etc. that can handle arrays containing a variable number of results.

Rgx.Case.AMATCH2, Rgx.Case.AMATCHES2 & Rgx.Case.AMATCHNTH functions

Search for values in one or more columns of a table, and return the relative position of the row(s) where a match is found.

Advanced MATCH functions returning:

- either the relative position of the first value found (AVMATCH2)
- or the relative position of all the values found (AVMATCHES2)
- or the relative position of the Nth value found (Rgx.Case.AMATCHNTH)

These functions are similar to the Rgx.AVLOOKUP2, Rgx.AVLOOKUPS2 and Rgx.AVLOOKUPNTH functions except that they return row numbers rather than values.

These functions are case-sensitive versions of Rgx.AMATCH2, Rgx.AMATCHES2 and Rgx.AMATCHNTH.

The Rgx.Case.AMATCH family of functions are multi-threaded, non-volatile array functions.

Rgx.Case.AMATCH Family Syntax

Rgx.Case.AMATCH2 (RegExp, Lookup_Table, Not_Found, Lookup_Columns, MemType_Name)

The first 3 parameters are required; the last 3 parameters are optional.

Rgx.Case.AMATCHES2 (RegExp, Lookup_Table, Lookup_Columns)

The first 2 parameters are required; the last parameter is optional.

Rgx.Case.AMATCHNTH (RegExp, Lookup_Table, Not_Found, Lookup_Columns, Position)

The first 2 parameters are required; the last 3 parameters are optional.

Rgx.Case.AMATCH2, Rgx.Case.AMATCHES2 and Rgx.Case.AMATCHNTH return row number(s) within the range specified by Lookup_Table.

RegExp (required)

Specifies the Regular Expression Pattern(s) to be matched against the value in the Lookup_Column(s).

Can be a single Regular Expression or multiple Regular Expressions arranged in columns (multiple lookup columns) and rows (multiple Lookup Rows).
A single Regular Expression can be a constant or a cell reference.
Multiple lookup Regular Expressione can be specified either as an array of constants or as a range referring to multiple cells. There should be the same number of columns of Regular Expression values as there are columns in the Lookup_Columns.

When doing multi-column lookups (multiple columns of both lookup values and lookup columns) the Rgx.Case.AMATCH functions will look for a row where ALL the lookup Regular Expression patterns are matched in the corresponding columns (Columns are treated as AND).

When doing multi-row lookups (multiple rows of lookup values) the Rgx.Case.AMATCH functions will look for a row separately for each row of lookup values (Rows are treated as OR).

A single Rgx.Case.AMATCH2 or Rgx.Case.AMATCHNTH statement will return the same number of rows and columns of result values as there are rows and columns in p.

Lookup_Table (required)

The rectangular range of cells or array or expression yielding an array to be used for the lookup table. The array/range must resolve to a single contiguous rectangular array.

Not_Found (optional, defaults to False)

Use this optional parameter to specify what to return if an exact match does not exist.

If the parameter is omitted the error value will be #N/A.

Lookup_Columns (optional, defaults to 1)

Use this optional parameter when you want to control which column(s) in Lookup_Table to use for the lookup.
The default value is 1: the first column in Lookup_Table

If the values are numeric they will be treated as column numbers, if alphabetic they will be treated as column labels to be found in the first row of Lookup_Table.

If there is more than one column given then there must be a matching set of values given in the columns of RegExp.

Multiple columns can be given either as an array of constants or as a range reference.

MemType_Name (Optional, Defaults to 2, Rgx.Case.AMATCH2 only)

Memory type for lookups can be 0, 1, 2 or 3 or a string that is used as the name for a memory.

- **3 = Global Memory for Rows or Columns** A single index is stored in global memory for each row (vertical lookups) or column (horizontal lookups).
  This global memory is super-efficient and easy to use when you have more than one lookup formula on the same row looking up different columns in a Table. This global memory works across all worksheets in all open workbooks.
- **2 = Book Sheet Row Memory** If you are using different Lookup Tables on different worksheets but still want to have more than one lookup formula on a row looking up different columns in the same table you can use this option which stores the row memory separately for each Workbook worksheet.
- **1 = Book Sheet Cell Memory** If you are using multiple Lookup Tables formulas in different cells on the same sheet and the same row you should use this option which stores the memory separately for each cell.
- **0 = Do not use Lookup memory**
- **Named Memory** If you give a text string instead of a number it will be used as a named memory. You can have many named memories in use at the same time (usually one for each Lookup Table). Each named memory works by row across all sheets within a workbook, so is the best choice when you have:
  - More than one memory lookup referring to different lookup tables within a single formula.
  - More than one memory lookup referring to different lookup tables in different cells on the same row, and these lookups are repeated on the same row in multiple sheets.

If this option is not zero, then Rgx.Case.AMATCH2 will first check to see if the index stored in memory that gave the answer the last time the Rgx.Case.AMATCH2 was calculated still gives the correct answer. If it does then Rgx.Case.AMATCH2 will return that answer without doing any more processing.
When using Lookup memory and there are multiple exact match answers available Rgx.Case.AMATCH2 will not necessarily return the answer from the same row as MATCH.

Rgx.Case.AMATCHES2 and Rgx.Case.AMATCHNTH do not use lookup memory.

Position (Optional, Defaults to 0, Rgx.Case.AMATCHNTH only)

Controls which result will be returned from multiple matches.

- N: where N is a positive integer. The Nth match found will be returned
- 0 : If sorted ascending the largest value that is less than or equal to RegExp
  If not sorted then the first value found
  If sorted descending then the smallest value that is greater than or equal to lookup value
- -1: The first value found will always be returned
- -2: the Last Value found will always be returned
- -3: All matches found will be returned

Remarks

Rgx.CaseAMATCHES2 with one or more rows of Lookup_Values, and **Rgx.Case.AMATCH2 or Rgx.Case.AMATCHNTH** with multiple rows of Lookup Values can return a variable number of rows depending on how many rows meet the lookup criteria. So you should use them either in a dynamic array formula that returns multiple rows or embedded in functions like SUM(), AVERAGE() etc. that can handle arrays containing a variable number of results.

EVAL2 function: evaluate a string formula

EVAL2 is an array function that evaluates a string as though it was an Excel formula or an Excel defined name, and returns the result of the evaluation, or #N/A if the string cannot be evaluated.

EVAL2 is NOT multi-threaded, is volatile and is an array function.

EVAL2Syntax

EVAL2 (theInput)

theInput

any mixture of constants, functions, expressions, names and ranges that evaluates to a string. If the resulting string cannot be evaluated as an excel formula or defined name the function will return #Value.

EVAL2 Remarks and Limits

EVAL2 is a volatile function.

Because EVAL2 is a volatile function every formula containing an EVAL2 function will be recalculated at every recalculation.

Unqualified cell references

The input string is evaluated as though it was on the worksheet in which the function has been entered. This means that unqualified cell references like F7 will be evaluated as referring to F7 on this sheet, regardless of which sheet is active. To refer to cells on other sheets use qualified references like Sheet3!F7.

Array Formulas

The input string is always evaluated as though it had been array-entered. An array of multiple results is returned where appropriate.

Date Literals

EVAL always interprets date literals as US style dates (MMDDYY).

Maximum 255 characters

EVAL2 will evaluate a maximum of 255 characters

Mathematical Functions

- LINTERP2D - Fast 2-dimensional linear interpolation
- VLINTERP2 - Fast linear interpolation in a vertical table
- GINICOEFF - Fast calculation of the Gini inequality coefficient.

VLINTERP2 function

VLINTERP2 provides efficient linear interpolation in a table of values.

The VLINTERP2 function is similar to VLOOKUP, except that it calculates linear interpolations between the values in a lookup table if an exact match cannot be found.

A single Lookup_Value and Col_Return_Nums will return a single interpolated value.
Multiple lookup values will return a column of values.
Multiple Col_Return_Nums will return a row of values.

VLINTERP2 is a multi-threaded, non-volatile array function.

VLINTERP2 Syntax

VLINTERP2(Lookup_Values, Table_Values, Col_Return_Nums, Lookup_ColNum, Extrapolate)

Lookup_Values

The value(s) to lookup in the Lookup_ColNum column of Table_Values. Multiple values will produce a column of results corresponding to the Lookup_Values.
If an exact match is found then VLINTERP2 will use that row.
If no exact match is found VLINTERP2 will interpolate between the row that contains the largest value that is smaller than lookup_row_value, and the row that contains the smallest value that is larger than lookup_row_value.

Table_Values

A rectangular set of values to be used for the interpolation. The first column must contain the values to be looked up. The other columns contain the values to be interpolated between.

Col_Return_Nums

The column number to return the interpolated values from, where the first column in Table_Values is 1. Multiple column numbers will produce a row of results corresponding to the requested columns.

Lookup_ColNum

Optional – Default 1. The column number to find the Lookup_Values from, where the first column in Table_Values is 1.

Extrapolate

Optional – True (default) to allow extrapolation beyond Table_Values

Calculation Method

If lookup_value1 and lookup_value2 are the two values found by the lookup in column 1, and interp_value1 and Interp_Value2 are the corresponding values from column Col_Return_Nums, then:

VLINTERP2=

interp_value1 + (interp_value2 – interp_value1) *
((Lookup_row_value-lookup_value1)/(lookup_value2-lookup_value1))

Remarks

Table_Values must be a rectangular set of values (range, array or expression) sorted by ascending on the Lookup column. It should not contain empty values in the Lookup column.

Table_Values should not contain column labels.

When using more than one Lookup_Value or Col_Return_Num VLINTERP2 should be entered as an array formula using Control/Shift/Enter.

VLINTERP2 returns #NA if:

- Extrapolate is FALSE and
  - Lookup_row_value is greater than the largest value in the first column of Table_Values.
  - Lookup_row_value is less than the smallest value in the first column of Table_Values..
- VLINTERP2 returns #VALUE if any of the used input values are not numeric, or if Lookup_ColNum is greater than the number of columns in Table_Values

LINTERP2D function

LINTERP2D provides efficient 2-dimensional linear interpolation in a table of values.

The LINTERP2D function is similar to VLOOKUP, except that it calculates 2-dimensional row and column linear interpolations between the values in a lookup table if an exact match cannot be found.

A single Lookup_1stCol and Lookup_1stRow will return a single interpolated value.

Multiple pairs of Lookup_1stCol and Lookup_1stRow values will return a vector of values.

LINTERP2D is a multi-threaded, non-volatile array function.

LINTERP2D Syntax

LINTERP2D (Lookup_1stCol, Table_Values, Lookup_1stRow)

Lookup_1stCol

the value(s) to lookup in the first column of Table_Values.

Table_Values

a rectangular set of values to be used for the interpolation. The first column must contain the values to be looked up using Lookup_1stCol. The first row must contain the values to be looked up using Lookup_1stRow. The other rows and columns contain the values to be interpolated between.

Lookup_1stRow

the value(s) to lookup in the first row of Table_Values.

Remarks

Table_Values must be a rectangular set of values (range, array or expression) sorted ascending on the first column and on the first row. It should not contain empty values in the first column or row.

Table_Values should not contain column or row labels.

When using more than one pair of Lookup_1stCol and Lookup_1stRow values LINTERP2D should be entered as an array formula using Control/Shift/Enter. Oe as a dynamic array formula

LINTERP2D returns #NA if Lookup_1stCol or Lookup_1stRow are outside the boundary values of Table_Values.

LINTERP2D returns #VALUE if any of the used input values are not numeric.

Calculating Gini Coefficients with GINICOEFF

This function is extremely efficient at calculating Gini coefficients on large data sets.

What are Gini Coefficients?

Gini Coefficients are a frequently-used method of measuring inequalities such as income distribution in a population.

A Gini Coefficient can be calculated as "the relative mean difference" - the mean of the difference between every possible pair of data points, divided by the mean of all the data points. A Gini Coefficient ranges from 0 (everyone has the same income) to 1 (one person has all the income).

Some Gini Income coefficients are:

- Sweden 0.23
- France 0.28
- UK 0.34
- USA 0.45
- Brazil 0.57
- Zimbabwe 0.57

(Source: http://en.wikipedia.org/wiki/List_of_countries_by_income_equality)

The Gini formula is often written as:

G=SUM(i=1 to n) SUM(j=1 to n) ABS(Data points(i)-Data points(j)) / (n*n*Average(Data points))
where Data points is the range of data and n is the number of points in Data points.
A Bias Correction factor of n/(n-1) is usually applied.

A more efficient formula has been developed by Angus Deaton (Princeton 1997):

G=(n+1)/(n-1)-2 /(n*(n-1)* Average(Data points))*SUM(i=1 to n) Data points(i)*Rank(i)
Where Rank is 1 for the largest value and n for the smallest value.
This formula has the bias correction factor built-in.

For more information on Gini Coefficients see:
http://en.wikipedia.org/wiki/Gini_coefficient or http://mathworld.wolfram.com/GINICOEFficient.html

GINICOEFF function

The GINICOEFF function calculates the Gini inequality coefficient using the Angus Deaton formula. Empty cells and blank cells are ignored. If there are any negative numbers in the input data the entire data series is offset by the absolute value of the largest negative number found. Bias correction is optional but applied by default.

GINICOEFF is multithreaded, is NOT volatile and is NOT an array function.

GINICOEFF Syntax:

GINICOEFF(InputData, Sorted, BiasCorrect)

InputData (required)

InputData must be a single row or column (range, array or calculated values) of numeric data.

Sorted (Optional)

- -1 for sorted descending
- 0 for unsorted
- 1 for sorted ascending
- The default is Unsorted

BiasCorrect (Optional)

- True to include the Bias Correction factor
- False to exclude the Bias Correction factor
- Default is True

IFERRORX Function

Use this function for trapping and handling errors in formulas. It is similar to the IFERROR function available in Excel 2007 and later Excel versions, but also works with earlier Excel versions and has options to pass through #N/A and #Value.

Description

Returns a value you choose if the formula expression returns an error, otherwise returns the result of the formula expression.

Sometimes you need to be able to trap all error values except #N/A or #VALUE so that the error value is allowed to propagate through dependent formulas. So IFERRORX allows you to optionally request that #N/A and/or #Value are NOT considered as errors by IFERRORX but are passed through instead.

Its more efficient to use IFERRORX instead of formulas like
=IF(ISERROR(VLOOKUP(A1,B1:C65000,2,FALSE)),"Not Found", VLOOKUP(A1,B1:C65000,2,FALSE))

IFERRORX is a multi-threaded, non-volatile array function.

IFERRORX Syntax

IFERRORX (TheExpression, Value_if_error, Exclude_NA, Exclude_Value)

TheExpression – Required

The value of formula expression to be checked for an error.

Value_if_error – Required

The value to return if Expression evaluates to an error.

Exclude_NA – Optional

If TRUE and the error is #N/A do NOT evaluate as an error and so return #N/A. Default FALSE.

Exclude_Value –Optional

If TRUE and the error is #VALUE do NOT evaluate as an error and so return #VALUE. Default FALSE.

Remarks

If the IFERRORX function is contained in an array formula and TheExpression returns an array of values then IFERRORX handles each value in the array individually.

If TheExpression evaluates to an empty cell IFERRORX treats it as a zero. This behaviour is not the same as the IFERROR function in Excel 2007 and later which treats an empty cell as an empty string.

The IFERRORX function is thread-safe.

Examples

=IFERRORX(VLOOKUP(A1,B1:C65000,2,FALSE),"Not Found") instead of
=IF(ISERROR(VLOOKUP(A1,B1:C65000,2,FALSE)),"Not Found", VLOOKUP(A1,B1:C65000,2,FALSE))

=IFERRORX(200/5,"Division Error") returns 40

=IFERRORX(200/0,"Division Error") returns "Division error"

=IFERRORX(MATCH(99,{1,2,3,4,5,6,7,8,99,100},0),0) returns 9 (the 9th position)

=IFERRORX(MATCH(99,{1,2,3,4,5,6,7,8,9},0),0) returns 0 (99 is not found so the #N/A is replaced by Value_if_error)

=IFERRORX(MATCH(99,{1,2,3,4,5,6,7,8,9},0),0,TRUE) returns #N/A (Exclude_NA is TRUE so the #N/A is not considered an error)

Reference Functions

- PREVIOUS - Returns the value of the cell at the previous calculation.
- SETMEM - Store the result of an expression to be retrieved later in the same formula
- GETMEM - Retrieve a result that was stored by SETMEM

PREVIOUS Function

PREVIOUS returns the value of the cell at the previous calculation.

PREVIOUS allows you to get the value from the previous calculation for the cell containing the PREVIOUS function.

Normally if you create a formula that refers to the cell that contains the formula, Excel detects a circular reference that requires using iteration to solve.

PREVIOUS does NOT cause a circular calculation.

You can use PREVIOUS as a parameter of a VBA UDF. This is useful when you have a slow UDF that you want to be able to short-circuit under certain conditions. For example, if your UDF retrieves data from a remote server you may only want to retrieve a new value when the server has been refreshed.

PREVIOUS is NOT thread-safe, is optionally volatile and does not handle arrays.

Previous Syntax

PREVIOUS (Volatile)

Volatile

Optional (default True). True causes the function to be calculated at every recalculation.

Do not use False for this argument when the formula containing the function uses or refers to a cell containing a volatile function.

Remarks and Limits

PREVIOUS is NOT a thread-safe function, and so any formula containing PREVIOUS will not take advantage of Excel's multi-threaded calculation.

Do not use False for the Volatile argument when the formula containing the function uses or refers to a cell containing a volatile function.

When a formula containing PREVIOUS is entered, the value from the previous recalculation will be reset.

Examples

=MyUDF(Refresh,theInput,PREVIOUS(FALSE))

If MyUDF is a VBA function that retrieves data from a remote server based on the parameters in theInput, you could program it to only get the data from the remote server when the Refresh switch was true, otherwise it could return the value from the last calculation.

=IF(A1>PREVIOUS(),A1,PREVIOUS())

Gives the largest value that cell A1 has ever contained.

=PREVIOUS(FALSE)+A1

Gives a cumulative sum of all the numbers that have changed in A1.

Note that if the same number is entered twice in succession in cell A1 this formula will NOT accumulate the same number twice.

=IF(COUNTA(C1)>0,PREVIOUS(FALSE)+1,0)

Counts the number of times the contents of cell C1 has been changed, reset to zero if C1 is cleared or the contents deleted.

Note that if the same value is entered twice in succession in cell A1 this formula will NOT increment the change count.

SETMEM and GETMEM Functions

Optimize your formulas by storing and retrieving the results of very calculation-intensive expressions used more than once in a formula.

Description

The SETMEM and GETMEM functions are thread-safe functions that you can use to avoid repeated calculation of expressions that occur more than once in the same formula.

Use SETMEM to store in memory the results of calculating a calculation-intensive formula expression so that you can replace the later repeats of the formula expression with GETMEM.

This can save time when the additional cost of using SETMEM and GETMEM is less than the cost of the repeated calculation of the calculation-intensive formula expression.

You can use GETMEM as many times as you like within a formula to retrieve the previously stored results.

SETMEM and GETMEM are both thread-safe, non-volatile, array functions

SETMEM Syntax

SETMEM (TheValue, SetID)

TheValue

The value, range, array, or expression result to be stored.

SetID (Optional. Default "A")

The ID (string) for the stored values. Default is "A". When you have more than one SETMEM within a single formula you should use a different SetID string for each SETMEM so that you can tell GETMEM which stored value to recall.

GETMEM Syntax

GETMEM (SetID)

SetID (Optional. Default "A")

The ID (string) for the previously stored temporary values. Default is "A".

Information Functions

Additional information functions:

- HASFORMULA2 - returns true if the cell contains a formula
- COUNTROWS2 and COUNTCOLS2 count the number of rows or columns from a referenced cell to the last visible non-empty cell in the row or column.
- COUNTCONTIGROWS2 and COUNTCONTIGCOLS2 count the number of rows or columns from a referenced cell to before the next visible empty cell.
- COUNTUSEDROWS2 and COUNTUSEDCOLS2 count the number of rows or columns from a referenced cell to the last cell in the used range.
- CALCSEQCOUNTREF, CALCSEQCOUNTSET, CALCSEQCOUNTVOL simplify the tracing of Excel's calculation sequence and determining if a function is volatile.

HASFORMULA2 function

The HASFORMULA function returns true if any of the cells in the range contain a formula.

HASFORMULA is NOT multi-threaded, NOT volatile and is NOT an array function.

HASFORMULA2 Syntax

HASFORMULA2(theRange)

theRange

Any valid range references.

The function returns true if any of the cells in the range contains a formula, otherwise false.

HASFORMULA 2Remarks

The function considers the following as formulas:

- =2
- +2
- =a1
- =”Text”
- =NOW()
- =RangeName

The function will return False for the following:

- 2
- 45
- “Text”

Calculation Sequence and Counting functions

SpeedTools contains three functions to help trace Excel's calculation sequence.
Two of these functions increment a counter, and the third allows you to reset the counter.

CALCSEQCOUNTREF Function

CalcSeqCountRef returns the value of the calculations counter, and is dependent on a cell reference, so you can use it to count the number of times a cell is calculated.

CalcSeqcountRef Syntax

CalcSeqCountRef (theRange)

theRange

A reference to an excel cell. At each recalculation Excel will calculate CalcSeqCountRef if the cell referred to by theRange has changed or been recalculated.

CalcSeqCountRef Remarks

The function uses a single global calculation counter, so a workbook containing multiple formulas containing CalcSeqCountRef will show the sequence in which they were calculated.
Entering CalcSeqCountRef into a cell using the Function Wizard/Palette will increment the calculation counter several times because the Function Wizard attempts to show you the result of the function at each step.
To reset the calculations counter either run the ZeroCalcSeq macro from Tools→Macros, or use the CalcSeqCountSet function.
See also the CalcSeqCountVol function, which is a volatile function which increments the same calculations counter.

CALCSEQCOUNTSET Function

This function allows you to set the calculation counter used by CalcSeqCountRef and CalcSeqCountVol to a value.

CalcSeqCountSet Syntax

CalcSeqCountSet (theInput)

TheInput

an integer value or a reference that can be resolved to an integer value.
This value is used to reset the calculation counter each time the function is executed. Alternatively, you can run the ZeroCalcSeq macro from Tools→Options, which will set the counter to zero.

CALCSEQCOUNTVOL function

This function is a volatile version of CalcSeqCountRef.

CalcSeqCountVol Syntax

CalcSeqCountVol ()

The function has no arguments and is volatile. The function will increment the calculation counter at each calculation or recalculation.

Functions for counting Rows and Columns

These functions give you the ability to find the number of rows or columns to the last empty cell, the next empty cell, or the last cell in the used range:

- COUNTROWS2 and COUNTCOLS2 count the number of rows or columns from a referenced cell to the last visible non-empty cell in the row or column.
- COUNTCONTIGROWS2 and COUNTCONTIGCOLS2 count the number of rows or columns from a referenced cell to before the next visible empty cell.
- COUNTUSEDROWS2 and COUNTUSEDCOLS2 count the number of rows or columns from a referenced cell to the last cell in the used range.

You can use these functions in Dynamic Range names and for the number of rows and columns arguments in OFFSET.

COUNTROWS2 and **COUNTCONTIGROWS2** can count rows in either a single column or multiple adjacent columns.

Similarly, **COUNTCOLS2** and **COUNTCONTIGCOLS2** can count columns in either a single row or multiple adjacent rows.

All these functions (COUNTROWS2, COUNTCOLS2, COUNTCONTIGROWS2, COUNTCONTIGCOLS2, COUNTUSEDROWS2 and COUNTUSEDCOLS2) are volatile functions.

Changing cell visibility using Automatic or Advanced filter will trigger a recalculation of COUNTROWS2, COUNTCOLS2, COUNTCONTIGROWS2 and COUNTCONTIGCOLS2.

Changing cell visibility using Hide, Unhide or by setting width or height to zero will NOT trigger a recalculation of COUNTROWS2, COUNTCOLS2, COUNTCONTIGROWS2 and COUNTCONTIGCOLS2.

These functions are volatile, NOT thread-safe, NOT array-capable functions.

COUNTROWS2 Function

The COUNTROWS2 function counts the number of rows from a reference row to the last **visible** non-empty cell in the referenced columns.

COUNTROWS2 Syntax

COUNTROWS2(theColumns)

TheColumns

A range reference to a cell or cells in the columns whose rows are to be counted.

COUNTROWS2 counts the number of rows, including blanks and empty cells down to the last **visible** non-empty cell.
The count includes the top row of the referenced cell.
If theColumns refers to more than one adjacent column then the count gives the largest row count found for the columns.

COUNTROWS 2Remarks

COUNTROWS 2is a volatile function.

If the reference specified for theColumns contains more than one row, the top row will be used.

If the reference specified for theColumns contains more than one column, the last **visible** non-empty row in each column will be found, and the largest row count will be returned.

COUNTROWS2uses the same criteria as Ctrl-Up Arrow to detect empty cells. The last row found is equivalent to selecting the last row in the worksheet and pressing Ctrl-Up-Arrow, except that it will stop at the cell specified by theColumns.

COUNTROWS2 will return 1 if the cell specified by theColumns is empty and there are no other **visible** non-empty cells below theColumns

A cell that contains nothing but is formatted is treated as an empty cell.

A cell that contains ‘ will look empty but will be treated as a non-empty cell.

Invisible cells can be created using Automatic or Advanced Filter, hiding rows or columns, or setting row heights or column widths to zero.

If the last row(s) in the range are not visible then COUNTROWS2 will not count them.
If intermediate row(s) in the range are not visible COUNTROWS2 will count them.

Countrows2should be used with care on ranges with hidden rows or rows with zero heights.	Because hiding rows/columns or setting width/heights to zero does not trigger a recalculation COUNTROWS2 may show an incorrect value even in automatic mode until the workbook is recalculated. Changing the Automatic or Advanced filter does trigger a recalculation.

COUNTROWS2 Example

Assuming that cell B55 contains 10 and there are no other non-empty cells in column B then:

=COUNTROWS2(B4) returns 52 (51 rows from B4 to B55 plus row 4 itself).

COUNTCONTIGROWS2 Function

The COUNTCONTIGROWS2 function counts the number of rows from a reference row down to the last cell before the next **visible** empty cell in the referenced columns.

COUNTCONTIGROWS2 Syntax

COUNTCONTIGROWS2 (theColumns)

TheColumns

A range reference to a cell or cells in the columns whose rows are to be counted.

COUNTCONTIGROWS2 counts the number of rows downwards from the referenced cell to the cell before the first **visible** empty cell found.
The count includes the top row of the referenced cell.
If theColumns refers to more than one adjacent column then the count gives the largest row count found for the columns.

COUNTCONTIGROWS2 Remarks

COUNTCONTIGROWS2 is a volatile function.

If the reference specified for theColumns contains more than one row, the top row will be used.

If the reference specified for theColumns contains more than one column, the last non-empty row before the first empty cell in each column will be found, and the largest rowcount will be returned.

COUNTCONTIGROWS2 uses the same criteria as Ctrl-Down Arrow to detect empty cells, except when the reference cell or the cell below the reference cell is empty.

COUNTCONTIGROWS2 will return 1 if the cell specified by theColumns is empty and the next cell down is also empty.

If the cell specified by theColumns is empty but the cells below it are not empty then COUNTCONTIGROWS2 will look downwards from the cell below the cell specified by theColumns.

A cell that contains nothing but is formatted is treated as an empty cell.

A cell that contains ‘ will look empty but will be treated as a non-empty cell.

Invisible cells can be created using Automatic or Advanced Filter, hiding rows or columns, or setting row heights or column widths to zero.

If intermediate empty row(s) in the range are not visible COUNTCONTIGROWS2 will NOT treat them as empty.

COUNTCONTIGROWS2 is not recommended for ranges with hidden rows or rows with zero heights.	Because hiding rows/columns or setting width/heights to zero does not trigger a recalculation COUNTCONTIGROWS2 may show an incorrect value even in automatic mode until the workbook is recalculated. Changing the Automatic or Advanced filter does trigger a recalculation. **COUNTCONTIGROWS2 is not recommended for ranges containing hidden rows or rows with zero heights.**

COUNTCONTIGROWS2 Example

Assuming that cells B4:B55 contain 10 and cell B2:B3 and cell B56 are empty then:

=COUNTCONTIGROWS2(B4) returns 52 (51 consecutive non-empty rows from B5 to B55 plus row 4 itself).

=COUNTCONTIGROWS2(B2) returns 1 (the cell below B2 is empty)

=COUNTCONTIGROWS2(B3) returns 53 (52 consecutive non-empty rows from B4 to B55 plus row 3 itself even though B3 is empty).

COUNTUSEDROWS2 Function

The COUNTUSEDROWS2 function counts the number of rows down to the end of the used range for this sheet.

COUNTUSEDROWS2 Syntax

COUNTUSEDROWS2 (theRow)

TheRow

A range reference to a row.

COUNTUSEDROWS2 counts the number of rows, including blanks and empty cells down to the end of the used range for the sheet containing theRow.
The count includes the top row of the referenced range.

COUNTUSEDROWS2 Remarks

COUNTUSEDROWS2 is a volatile function.

If the reference specified for theRow contains more than one row, the top row will be used.

The used range may contain empty rows or columns.

Whenever the COUNTUSEDROWS2 function is calculated the used range for the worksheet containing theRow will be reset.

COUNTUSEDROWS2 Example

Assuming that cell B55 contains 10 and cell Z75 is empty but coloured yellow , and no other cells have been formatted then:

=COUNTUSEDROWS2(B4) returns 72 (row 75 is the last row in the used range less rows 1-3).

COUNTCOLS2 Function

The COUNTCOLS2 function counts the number of columns before the last **visible** non-empty cell in the referenced row.

COUNTCOLS2 Syntax

COUNTCOLS2(theRows)

TheRows

A range reference to a cell or cells in the rows whose columns are to be counted.

COUNTCOLS2 counts the number of columns, including blanks and empty cells across to the right to the last **visible** non-empty cell.
The count includes the first column of the referenced cell.
If theRows refers to more than one adjacent row then the count gives the largest column count found for the rows.

COUNTCOLS2 Remarks

COUNTCOLS2 is a volatile function.

If the reference specified for theRows contains more than one column, the first column will be used.

If the reference specified for theRows contains more than one row, the last non-empty column in each row will be found, and the largest column count will be returned.

COUNTCOLS2 uses the same criteria as Ctrl-Left Arrow to detect empty cells. The last column found is equivalent to selecting the last column in the worksheet and pressing Ctrl-Left, except that it will stop at the cell specified by theRows.

COUNTCOLS2 will return 1 if the cell specified by theRows is empty and there are no other non-empty cells to the right of theRows

A cell that contains nothing but is formatted is treated as an empty cell.

A cell that contains ‘ will look empty but will be treated as a non-empty cell.

Invisible cells can be created using Automatic or Advanced Filter, hiding rows or columns, or setting row heights or column widths to zero.

If the last column(s) in the range are not visible COUNTCOLS2 **will not count** them.

If intermediate column(s) in the range are not visible COUNTCOLS2 **will count** them.

COUNTCOLS2 should be used with care on ranges with hidden rows or rows with zero heights.	Because hiding rows/columns or setting width/heights to zero does not trigger a recalculation COUNTCOLS2 may show an incorrect value even in automatic mode until the workbook is recalculated. Changing the Automatic or Advanced filter does trigger a recalculation.

COUNTCOLS2 Example

Assuming that cell D42 contains FRED and there are no other non-empty cells in row 42 then:

=COUNTCOLS2(B42) returns 3 (2 columns from B42 to D42 plus the B column itself).

COUNTCONTIGCOLS2 Function

The COUNTCONTIGCOLS2 function counts the number of COLUMNS from a reference row down to the last cell before the next **visible** empty cell in the referenced columns.

COUNTCONTIGCOLS2 Syntax

COUNTCONTIGCOLS2 (theRows)

TheRows

A range reference to a cell or cells in the rows whose columns are to be counted.

COUNTCONTIGCOLS2 counts the number of columns across to the right from the referenced cell to the cell before the first **visible** empty cell found.
The count includes the first column of the referenced cell.
If theRows refers to more than one adjacent row then the count gives the largest column count found for the rows.

COUNTCONTIGCOLS2 Remarks

COUNTCONTIGCOLS2 is a volatile function.

If the reference specified for theRows contains more than one column, the left-most column will be used.

If the reference specified for theRows contains more than one row, the last non-empty column before the first empty cell in each row will be found, and the largest column count will be returned.

COUNTCONTIGCOLS2 uses the same criteria as Ctrl-Right Arrow to detect empty cells, except when the reference cell or the cell to the right of the reference cell is empty.

COUNTCONTIGCOLS2 will return 1 if the cell specified by theRows is empty and the next cell to the right is also empty.

If the cell specified by theRows is empty but the cells to the right of it are not empty then COUNTCONTIGCOLS2 will look to the right from the cell adjacent to the cell specified by theRows.

A cell that contains nothing but is formatted is treated as an empty cell.

A cell that contains ‘ will look empty but will be treated as a non-empty cell.

Invisible cells can be created using Automatic or Advanced Filter, hiding rows or columns, or setting row heights or column widths to zero.

If intermediate empty column(s) in the range are not visible COUNTCONTIGCOLS2 will NOT treat them as empty.

COUNTCONTIGCOLS2 is not recommended for ranges with hidden columns or columns with zero heights.

Because hiding rows/columns or setting width/heights to zero does not trigger a recalculation COUNTCONTIGCOLS2 may show an incorrect value even in automatic mode until the workbook is recalculated. Changing the Automatic or Advanced filter does trigger a recalculation.

COUNTCONTIGCOLS2 is not recommended for ranges containing hidden columns or columns with zero heights.

COUNTCONTIGCOLS2 Example

Assuming that cells C4:G4 contain 10 and cell A4, B4 and H4 are empty then:

=COUNTCONTIGCOLS2(C4) returns 5 (4 consecutive non-empty COLUMNS from D4 to G4 plus column C itself).

=COUNTCONTIGCOLS2(A4) returns 1 (the cell to the right of A4 is empty)

=COUNTCONTIGCOLS2(B4) returns 6 (5 consecutive non-empty COLUMNS from C4 to G4 plus column B itself even though B4 is empty).

COUNTUSEDCOLS2 Function

The COUNTUSEDCOLS2 function counts the number of columns across right to the end of the used range for this sheet.

COUNTUSEDCOLS2 Syntax

COUNTUSEDCOLS2 (theColumn)

TheColumn

A range reference to a cell in a column.

COUNTUSEDCOLS2 counts the number of columns, including blanks and empty cells across right to the end of the used range for the sheet containing theColumn.
The count includes the leftmost column of the referenced range.

COUNTUSEDCOLS2 Remarks

COUNTUSEDCOLS2 is a volatile function.

If the reference specified for theColumn contains more than one column, the leftmost row will be used.

The used range may contain empty rows or columns.

Whenever the COUNTUSEDCOLS2 function is calculated the used range for the worksheet containing theColumn will be reset.

COUNTUSEDCOLS2 Example

Assuming that cell C55 contains 10 and cell Z100 is empty but coloured yellow, and no other cells have been formatted then:

=COUNTUSEDCOLS2(C4) returns 24 (column Z is the last column (26) in the used range less columns A and B).

Examples and comparison of the counting functions

	A	B
1		
2		Sales
3		23/08/90
4		
5		19
6		09/08/03 10:50
7		TRUE
8		#DIV/0!
9		
10		

Assuming that the whole of column A and rows 11 to 65536 are empty, using the counting functions on the ranges B1:B10 and A1:A10 gives these results:

COUNTA counts all non-empty cells:
=COUNTA(B1:B10) gives 6
=COUNTA(A1:A10) gives 0

COUNT counts all numbers:
=COUNT(B1:B10) gives 3
=COUNT(A1:A10) gives 0

COUNTROWS2 counts the number of rows before the last empty cell: =COUNTROWS2(B1) gives 8
=COUNTROWS2(A1) gives 1
(always counts the referenced cell even if it is empty).

COUNTUSEDROWS2 counts the number of rows to the last row in the used range:
=COUNTUSEDROWS2(B1) gives 9
(B9 is empty but used because it is formatted)
=COUNTUSEDROWS2(A1) gives 9
(although column A is empty the last used range row is 9)

COUNTCONTIGROWS2 counts the number of rows before the next empty cell:
=COUNTCONTIGROWS2(B1) gives 3
(always counts the referenced cell even if it is empty)
=COUNTCONTIGROWS2(B5) gives 4
=COUNTCONTIGROWS2(A1) gives 1
(Always counts the referenced cell even if it is empty)

COUNTBLANK counts the number of blank or empty cells: =COUNTBLANK(B1:B10) gives 4
=COUNTBLANK(B1:B10) gives 10

COUNTROWS2 is faster than COUNTA when counting a single column.

Using the Count functions in dynamic range names

The usual technique of using COUNTA in dynamic range names is difficult or impossible to use if:

- The range contains empty cells.
- There are multiple ranges above one another on the same sheet.
- The columns in the range can contain a different number of rows.

If the range can contain empty cells then COUNTA will not give the number of rows or columns in the range. Using COUNTROWS2 will give the correct number of rows.

If you have multiple ranges stacked above one another on the same sheet but separated by blank rows it is difficult to use COUNTA. Using COUNTCONTIGROWS2 can help in this situation provided that each range does not itself contain blank cells.

If the columns in the range can contain a different number of rows one approach to getting the total number of rows in the range is to use COUNTUSEDROWS2, but this may overestimate the number of rows in the range, and will not work if you have vertically stacked ranges.

A better approach is to tell COUNTROWS2 or COUNTCONTIGROWS2 to look for the last row in multiple columns (theColumns refers to a range which is a single row spanning multiple columns). In this case the functions will return the largest row count found by evaluating each column in turn.

Similar remarks apply to the COUNTCOLS2 and COUNTCONTIGCOLS2 functions.

Text Functions

- CONCAT.RANGE - Concatenate cell values to a string using delimiters.
- PAD.TEXT - Expand/Contract text-strings to a fixed length
- REVERSE.TEXT - Reverse the characters in a Text-string
- SPLIT.TEXT - Splits text into tokens using delimiters
- GROUPS - Extracts groups of characters (digits, strings etc) from a Text-string
- Rgx.FIND - Find the position of a Regular Expression pattern within a text-string
- Rgx.LEN - Finds the length of the substring that matches a Regular Expression pattern
- Rgx.SUBSTITUTE - Replaces substring(s) that match Regular Expression patterns.
- Rgx.MID - Extracts substring(s) that match Regular Expression patterns.
- COMPARE - Compares 2 values using the same collating sequence as Excel's SORT
- ISLIKE2 - Compares a string to a wild-card expression
- Rgx.ISLIKE - Compares a string to a regular expression pattern

CONCAT.RANGE – concatenate range data

Concatenate cell values to a string using optional string delimiters. You can add strings to the start and end, format numbers and skip or replace blank/empty cells.

CONCAT.RANGE is a multithreaded, non-volatile, non-array function.

CONCAT.RANGE Syntax

CONCAT.RANGE (ConcatThis, Divider, NumberFormat, BlankSkipReplace, LineStart, LineEnd)

ConcatThis

A range or array of values to be concatenated.

Divider (optional, default "")

The character(s) to use as a divider between the concatenated items

NumberFormat (optional)

The number format to apply to numeric values.

BlankSkipReplace (optional, default False)

- If True then empty and zero-length values will be ignored.
- If False then empty and zero-length values will be included.
- If a Text-string it will be used to replace empty and zero-length values.

LineStart (optional, default "")

Text-string to insert before the first value.

LineEnd (optional, default "")

Text-string to insert after the last value.

PAD.TEXT function

Expands (or contracts) a text-string (or range/array of text strings) with one or more repeated Pad characters. Pad characters can be added to the start or end (PadEnd=True) of the text-string. PAD.TEXT is a multi-threaded, non-volatile array function.

Based on an idea by Kevin (Zorvek) Jones

PAD.TEXT Syntax

PAD.TEXT (Pad, Text, Length, PadEnd)

Pad (Optional, default space)

The characters to use for padding the text string, repeated as required.

Text (Required)

The Text-string(s) to be padded. Can be a range or an array of text-strings.

Length (Required)

The length you want for the output text after padding

PadEnd (Optional, default False)

If False pad the front of the Text-string, if True then pad the end of the Text-string.
When padding the front of the text string characters are taken from the right of the Pad text.
When padding the back of the text-string characters are taken from the left of the Pad text.

PAD.TEXT Examples

PAD.TEXT("X","ABC",5) = XXABC
PAD.TEXT("XYZ","ABC",5) = YZABC
PAD.TEXT("XYZ","ABC",5,TRUE) = ABCXY
PAD.TEXT(,"ABCDEF",5,TRUE) = ABCDE
PAD.TEXT(,"ABCDEF",5,FALSE) = BCDEF
PAD.TEXT("xy","AB",5) = yxyAB
PAD.TEXT("xy","AB",5,TRUE)= ABxyx

REVERSE.TEXT Function

Returns reversed Text from a single Text or array/Range of Texts: “abcdE” becomes Edcba”. Values that are not text are returned unchanged.
REVERSE.TEXT is a multi-threaded, non-volatile array function.

REVERSE.TEXT Syntax

REVERSE.TEXT (Text, Num_chars)

Text (required)

An expression, range or array containing one or more text-strings to be reversed

Num_chars (optional, default 0)

Num_chars gives the number of characters to be reversed.
0 means reverse all the characters.
A positive number n returns the reverse of the last n characters only.
A negative number –n returns the reverse of the first n characters only.

Example

REVERSE.TEXT(“abcDEF”)=”FEDcba”
REVERSE.TEXT(“abcDEF”,3)=”FED”
REVERSE.TEXT(“abcDEF”,-3)=”cba”

SPLIT.TEXT Function

Splits a text string or a vertical array/range of strings into horizontal arrays of string tokens, using a string of delimiters.
Each individual character in the delimiters string is used as an alternative delimiter.
Alphabetic characters used as delimiters are case-sensitive.
SPLIT.TEXT is a multi-threaded, non-volatile array function.

When using SPLIT.TEXT on a vertical array/range of strings a rectangular array is returned where the number of columns is the maximum number of tokens returned by any row in the vertical array/range. Rows where the number of tokens returned is less than the maximum number of tokens returned across all rows will be padded out with empty string tokens.

SPLIT.TEXT Syntax

SPLIT.TEXT (Text, Delimiters, Count, Compress)

Text (Required)

The text string or vertical array/range of strings to be split

Delimiters

The list of delimiters to search for in the text string. Note that these delimiters are case-sensitive.

Count (Optional, default 0)

Integer. The position/number of string-tokens to return.

- 0 means return all tokens
- +n means return the n'th token
- -n means return the first n tokens
- If Count is larger than the number of tokens for a row then an empty string is returned.

Compress (Optional, default TRUE)

If TRUE empty string tokens will be ignored.

GROUPS Function

Extracts group(s) of adjacent characters from a text-string or an array/range of text-strings.
The characters to include in the group are defined by GroupType. GroupType can either be a Regular Expression pattern or a number for pre-defined group types:

- 0 – Numeric Digits (0-9)
- 1 – Alphabetic characters (a-zA-Z)
- 2 – All characters except numeric digits 0-9
- 3 – Numeric Digits 0-9 and the decimal separator character
- 4 – All characters except numeric digits 0-9 and the decimal separator character

You can choose either to select all groups or the nth group from the start or end of the text-string.
You can set the maximum number of characters to be returned, either from the left or the right of the result set of characters.
You can also control the start and end point of the search for groups within the text-string.
GROUPS is a multi-threaded, non-volatile array function.

GROUPS Syntax

GROUPS (Text, GroupNumber, MaxChars, GroupType, StartPos, EndPos)

Text (Required)

The text to extract the groups of characters from. Can be a range or array or a constant or any expression that returns a string/strings..

GroupNumber (Optional, default 1)

The position number of the group to be extracted from within the text-string.

- A value of zero extracts all the groups within the text-string and concatenates all the groups of characters.
- A positive number extracts the Nth group from the start of the text-string, working forwards.
- A negative number extracts the Nth group from the end of the text-string, working backwards.

MaxChars (Optional, default 0)

The maximum number of characters to be returned. A value of zero returns all the extracted characters. A positive number restricts the number of characters starting from the left: for example 3 would return only at most the 3 leftmost characters from the extracted characters. A negative number restricts the number of characters starting from the right, so a value of -3 would return only at most the 3 rightmost characters from the extracted characters.

GroupType (Optional, default 0)

Defines the type of characters to be considered as part of a group and returned. If GroupType is a string it will be treated as a regular expression pattern that defines which characters that will be considered part of a group. If GroupType is a number it will select one of the pre-defined Regular Expression patterns.

Any characters not defined as part of a group will act as separator characters between groups.

The numbers for the pre-defined patterns, with their equivalent Regular Expression patterns, are:

- 0 – (Default) Groups of adjacent numeric characters (0-9) will be returned: "[0-9]"
- 1 – Groups of adjacent alphabetic characters (a-z and A-Z) will be returned: "[a-zA-Z]"
- 2 – Groups of adjacent non-numeric characters will be returned: "[^0-9]"
- 3 – Groups of adjacent numeric characters (0-9) and the decimal separator will be returned: "[0-9.]"
- 4 – All adjacent characters except numeric characters (0-9) and the decimal separator will be returned: "[^0-9.]"

StartPos (Optional, default 1)

The position of the first character within Text to be considered as potentially part of a group. The position of the first character in Text is 1.

EndPos (Optional, default 0)

The position of the last character within Text to be considered as potentially part of a group. 0 signifies the last character in Text. The position of the first character in Text is 1.

Use StartPos and EndPos to limit the part of Text to be searched for groups.

GROUPS Examples:

	A	B	C	D	E	F
1	**Examples of the GROUPS Function**					
2						
3	123_abc45.zz06x	crw4xyz9955		123.4567		123.4567
4						
5	**GroupType Examples**					
6	GROUPS($A$3:$B$3,0)	1234506	49955			
7	GROUPS($A$3:$B$3,0,,1)	abczzx	crwxyz			
8	GROUPS($A$3:$B$3,0,,2)	_abc.zzx	crwxyz			
9	GROUPS($A$3:$B$3,0,,3)	12345.06	49955			
10	GROUPS($A$3:$B$3,0,,4)	_abczzx	crwxyz			
11	GROUPS($A$3:$B$3,0,,"[xyzb]")	bzzx	xyz			
12	**GroupNumber Examples**					
13	GROUPS($A$3:$B$3,2)	45	9955			
14	GROUPS($A$3:$B$3,2,,1)	zz	xyz			
15	GROUPS($A$3:$B$3,1,,2)	_abc	crw			
16	GROUPS($A$3:$B$3,-1,,1)	x	xyz			
17	GROUPS($A$3:$B$3,-1)	06	9955			
18	GROUPS($A$3:$B$3,0,3)	123	499			
19	GROUPS($D$3,1)	123				
20	GROUPS($D$3,2)	4567				
21	GROUPS($F$3)	123				
22	**MaxChars Examples**					
23	GROUPS($A$3:$B$3,0,-4)	4506	9955			
24	GROUPS($A$3:$B$3,1,-2)	23	4			
25	GROUPS($A$3:$B$3,1,99)	123	4			
26	**StartPos and EndPos Examples**					
27	GROUPS($A$3:$B$3,0,0,0,9)	506	955			
28	GROUPS($A$3:$B$3,0,0,0,9,13)	50	955			

Rgx.FIND function

Searches a string (or a rectangular array/range) for a substring that matches a regular expression pattern (or a rectangular array/range of patterns).

Returns the position (or array of positions) of the Nth substring matching the Regular Expression pattern. Rgx.FIND is a multi-threaded, non-volatile array function.

Rgx.FIND Syntax

Rgx.FIND(String, RegExp, Nth, Case_Sensitive)

The first 2 parameters are required; the last 2 parameters are optional.

String (required)

Constant, range or array containing the string(s) to be searched for the Regular Expression Patterns.

RegExp (required)

Specifies the Regular Expression(s) to be used when matching String. If String is an array or a multi-cell range then RegExp must contain a matching set (same number of rows and columns as String) of regular expressions

Nth (optional – default 1)

The start position of the Nth match of the substring is returned.

Case_Sensitive (optional)

TRUE to make the pattern-matching case-sensitive. The default is FALSE.

Rgx.FIND Examples

	A	B
34	**Start position of match**	**Rgx.Find**
35		\b5[1-5][0-9]{14}\b
36		**=Rgx.FIND($B$35,A37)**
37	Mastercard number 5499000001234567 ?	19
38	Is This a Mastercard number 5199000001234567 ?	29
39	Is This a Mastercard number 5699000001234567 ?	0

Rgx.LEN function

Searches a string (or a rectangular array/range) for a substring that matches a regular expression patter (or a rectangular array/range of patterns).

Returns the length (or array of lengths) of the Nth substring matching the Regular Expression pattern. Rgx.LEN is a multi-threaded, non-volatile array function.

Rgx.LEN Syntax

Rgx.FIND(String, RegExp, Nth, Case_Sensitive)

The first 2 parameters are required; the last 2 parameters are optional.

String (required)

Constant, range or array containing the string(s) to be searched for the Regular Expression Patterns.

RegExp (required)

Specifies the Regular Expression(s) to be used when matching String. If String is an array or a multi-cell range then RegExp must contain a matching set (same number of rows and columns as String) of regular expressions

Nth (optional – default 1)

The length of the Nth match of the substring is returned.

Case_Sensitive (optional)

TRUE to make the pattern-matching case-sensitive. The default is FALSE.

Rgx.LEN Examples

	A	B
27	**Number of characters matching**	**Rgx.Len**
28		\b5[1-5][0-9]{14}\b
29		**=Rgx.LEN($B$28,A30)**
30	Mastercard number 5499000001234567 ?	16
31	Is This a Mastercard number 5199000001234567 ?	16
32	Is This a Mastercard number 5699000001234567 ?	0

Rgx.SUBSTITUTE function

Searches a string (or a rectangular array/range) for a substring that matches a regular expression pattern (or a rectangular array/range of patterns).

Replaces the matched substring(s) with New_Text.
Rgx.SUBSTITUTE is a multi-threaded, non-volatile array function.

Rgx.SUBSTITUTE Syntax

Rgx.SUBSTITUTE(Text, RegExp, New_Text, Nth, Case_Sensitive)

The first 2 parameters are required; the last 3 parameters are optional.

Text (required)

Constant, range or array containing the string(s) to be searched for the Regular Expression Patterns.

RegExp (required)

Specifies the Regular Expression(s) to be used when matching String. If String is an array or a multi-cell range then RegExp must contain a matching set (same number of rows and columns as String) of regular expressions

New_Text (optional – default 0)

The Text(s) used to replace the substring(s) that match the Regular Expression Pattern. If omitted the matching substrings are removed.

Nth (optional – default 0)

If >0 replaces only the Nth match of the substring. If 0 all matches found are replaced.

Case_Sensitive (optional)

TRUE to make the pattern-matching case-sensitive. The default is FALSE.

Rgx.SUBSTITUTE Examples

	A	B
48	**Replace the Matching string**	**Rgx.SUBSTITUTE**
49		\b5[1-5][0-9]{14}\b
50		**=Rgx.SUBSTITUTE(A51,$B$49,"XXXX")**
51	Mastercard number 5499000001234567 ?	Mastercard number XXXX ?
52	Is This a Mastercard number 5199000001234567 ?	Is This a Mastercard number XXXX ?
53	Is This a Mastercard number 5699000001234567 ?	Is This a Mastercard number 5699000001234567 ?

Rgx.MID function

Searches a string (or a rectangular array/range) for a substring that matches a regular expression patter (or a rectangular array/range of patterns).

Returns the Nth substring that matches the Regular Expression pattern.
Rgx.MID is a multi-threaded, non-volatile array function.

Rgx.MID Syntax

Rgx.MID (String, RegExp, Nth, Case_Sensitive)

The first 2 parameters are required; the last 2 parameters are optional.

String (required)

Constant, range or array containing the string(s) to be searched for the Regular Expression Patterns.

RegExp (required)

Specifies the Regular Expression(s) to be used when matching String. If String is an array or a multi-cell range then RegExp must contain a matching set (same number of rows and columns as String) of regular expressions

Nth (optional – default 1)

The Nth matching substring is returned.

A value of zero returns a horizontal array (row) containing all the matching substrings. In this case only a single string and single corresponding RegExp is allowed.

Case_Sensitive (optional)

TRUE to make the pattern-matching case-sensitive. The default is FALSE.

Rgx.MID Examples

	A	B
41	**Extract the Matching string**	**Rgx.Mid**
42		\b5[1-5][0-9]{14}\b
43		**=Rgx.MID(A44,$B$42)**
44	Mastercard number 5499000001234567 ?	5499000001234567
45	Is This a Mastercard number 5199000001234567 ?	5199000001234567
46	Is This a Mastercard number 5699000001234567 ?	

COMPARE function

Compares 2 values using the same collating sequence as Excel's SORT.
Returns -1 if the first value is less than the second value, 0 if they are equal and +1 if the first value is greater than the second value.

COMPARE Syntax

COMPARE (Value1, Value2, Case_Sensitive)

Value1 (required)

The Value to compare with Value 2. Can be a string, a number or a logical value.

Value2 (required)

The Value to compare with Value 1. Can be a string, a number or a logical value.

Case-Sensitive (optional, default False)

If TRUE then the comparison of 2 Text values will be done in a case-sensitive manner.

Remarks

Returns #Value if either of Value1 or Value 2 are errors.

Returns a negative value if Value1 is less than Value2

Returns zero if Value1 is equal to Value2

Returns a positive value if Value1 is greater than Value2

The comparison is done using Excel's sorting sequence rules:

- Numbers <text <logical
- Lower-case characters<upper-case characters
- Numbers stored as text are always treated as text rather than numbers
- The comparison of characters and some character pairs is dependent on the Locale collating sequence in force when the Excel session was started.

ISLIKE2 array function for pattern-matching strings

The ISLIKE function performs a wild-card comparison between a string and a pattern using VBA Like, and returns True if the string matches the pattern, or False if it does not or an error occurs.

ISLIKE can be used as an array function

ISLIKE is multithreaded, is non-volatile and is an array function.

ISLIK2E Syntax

ISLIKE2 (theString, thePattern, CaseSensitive)

theString

An expression, constant or Range that can be resolved to a string or array of strings: the string(s) to be compared to the pattern. If theString contains numeric values they will be converted to strings.

ThePattern

An expression, constant or Range that can be resolved to a string or arrays of strings: the pattern(s) to be used in the comparison.

Patterns can contain:

- * Any number of characters, including none
- ? Any single character

ISLIKE2 Array function usage

When ISLIKE2 is used as an array function or inside an array formula each of the input arguments can be a single value, a one or two-dimensional array or a one or two-dimensional range.

If the dimensions of the two input arguments do not match, the smaller dimension will be wrapped: for example

=ISLIKE2(“Fred7”,{”*3”,”*5”,”*7”}) compares Fred7 to each of *3, *5, *7 in turn and returns {False,False,True}

ISLIKE2 Examples

True =ISLIKE2("aBBBa" ,"a*a")

True =ISLIKE2("BAT123khg","B?T*")

False =ISLIKE2("CAT123khg","B?T*")

Rgx.ISLIKE function

Tests whether a string matches a Regular Expression Pattern.

Rgx.ISLIKE is a multi-threaded, non-volatile array function.

Rgx.ISLIKE Syntax

Rgx.ISLIKE(String, RegExp, Case_Sensitive)

The first 2 parameters are required; the last parameter is optional.

String (required)

Text to be matched by the Regular Expression Pattern.

RegExp (required)

Specifies the Regular Expression to be used when matching the String.

Case_Sensitive (optional)

TRUE to make the pattern-matching case-sensitive. The default is FALSE.

Rgx.ISLIKE Examples

	A	B
1		**Rgx.ISLIKE**
2	**Emails**	^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$
3		**=Rgx.ISLIKE(A4,$B$2)**
4	Charles@DecisionModels.com	TRUE
5	abcd.xyz@fred.co.uk	TRUE
6		
7	**MasterCard Number**	^5[1-5][0-9]{14}$
8		**=Rgx.ISLIKE(A9,$B$7)**
9	5499000001234567	TRUE
10	5199000001234567	TRUE
11	5699000001234567	FALSE
12	549900000123456	FALSE
13		
14	**MasterCard number in the String**	\b5[1-5][0-9]{14}\b
15		**=Rgx.ISLIKE($A16,$B$14,,TRUE)**
16	Mastercard number 5499000001234567 ?	TRUE
17	Is This a Mastercard number 5199000001234567 ?	TRUE
18	Is This a Mastercard number 5699000001234567 ?	FALSE

	A	B
55	**Telephone Numbers**	**Rgx.ISLIKE**
56		1?\W*([2-9][0-8][0-9])\W*([2-9][0-9]{2})\W*([0-9]{4})(\se?x?t?(\d*))?
57		**=Rgx.ISLIKE(A58,$B$56)**
58	1-234-567-8901	TRUE
59	1-234-567-8901 x1235	TRUE
60	1-234-567-8901 Ext1234	TRUE
61	1 (234) 567-8901	TRUE
62	1.234.567.8901	TRUE
63	1/234/567/8901	TRUE
64	12345678901	TRUE

33			RegEx Pattern
34			1?\W*([2-9][0-8][0-9])\W*([2-9][0-9]{2})\W*([0-9]{4})(\se?x?t?(\d*))?
35	**Numbers**		**Rgx.ISLIKE(A36,$C$34)**
36	1-234-567-8901		TRUE
37	1-234-567-8901 x1234		TRUE
38	1-234-567-8901 ext1234		TRUE
39	1 (234) 567-8901		TRUE
40	1.234.567.8901		TRUE
41	1/234/567/8901		TRUE
42	12345678901		FALSE

The Numbers are in A36:A42, stored as text.

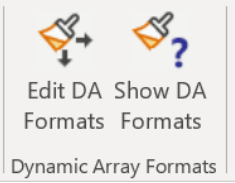
The Regular Expression Pattern is in C34.

The Rgx.ISLIKE formula is copied down in each cell of C36:C42.

You can also use Rgx.ISLIKE as an array formula

{=Rgx.ISLIKE(A36:A42,C34)} would give the same results as the 7 individual formulas.

Formatting Dynamic Arrays

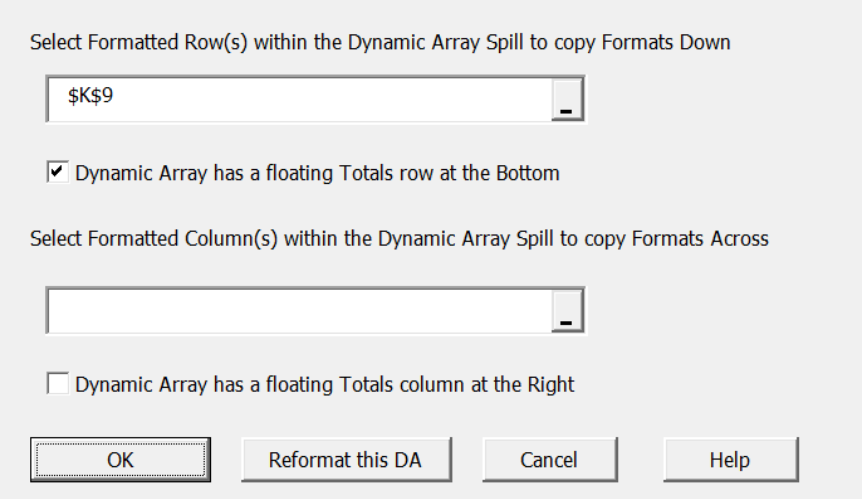


SpeedTools dynamic array formats provide extensive control over the formatting of resizing dynamic arrays.

- Format the dynamic array the way you want using Excel's formatting tools.
- Specify rows and columns as format templates whose formats you want to repeat.
- Use different formatting for floating row and column totals.
- Use conditional formatting rules.

Edit DA Formats

Select any cell in a formatted Dynamic Array and click Edit DA Formats.



If the dynamic array expands/contracts vertically select the formatted row(s) you want to repeat copy down and check the floating Row Totals box if required.

If the dynamic array expands/contracts horizontally select the formatted column(s) you want to repeat copy across and check the floating Column Totals box if required.

Filters:	Mississippi		0
Yield:	51.52		

Invoice	Product	Region	Units	Price	Revenue
57	Digital Projector	Mississippi	5	86.830	434.15
284	Digital Projector	Mississippi	5	84.240	421.20
312	Digital Projector	Mississippi	3	87.970	263.91
4	Home Automation Hub	Mississippi	5	24.700	123.50
353	Home Automation Hub	Mississippi	5	22.050	110.25
158	Home Automation Hub	Mississippi	4	23.690	94.76
69	Digital Projector	Mississippi	1	85.110	85.11
403	Home Automation Hub	Mississippi	2	21.380	42.76
318	Home Automation Hub	Mississippi	1	21.440	21.44
			31		**1597.08**

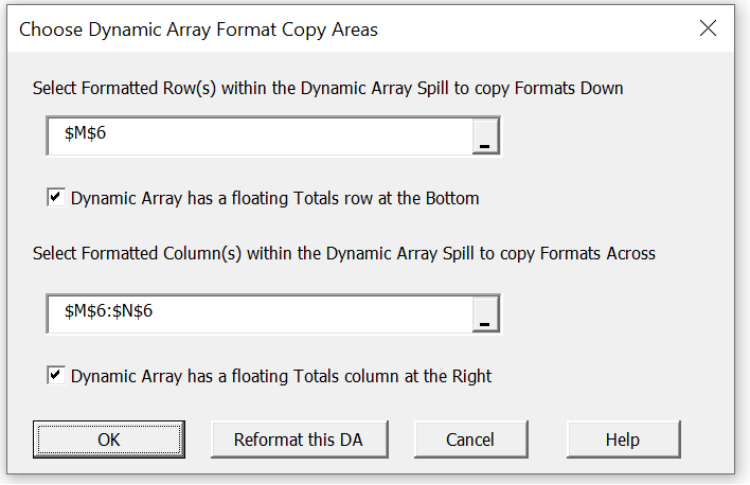
In this example the dynamic array only expands/contracts vertically and the $K$9 is a cell in the first row. This tells SpeedTools to use the first row as a formatting template for the other rows.
Because the dynamic array has a floating totals row which is formatted differently the Floating Row Totals box is checked.
Changing the filter to 200 contracts the dynamic array and it is dynamically reformatted.

Filters:	Nebraska		300
Yield:	85.56		

Invoice	Product	Region	Units	Price	Revenue
68	Digital Projector	Nebraska	5	85.940	429.70
70	Digital Projector	Nebraska	4	85.570	342.28
287	Digital Projector	Nebraska	4	85.060	340.24
			13		**1112.22**

You can use more than one row and/or column as the formatting template to repeat:

1	2	3	4	5	6	**21**
7	8	9	10	11	12	**57**
13	14	15	16	17	18	**93**
19	20	21	22	23	24	**129**
25	26	27	28	29	30	**165**
31	32	33	34	35	36	**201**
96	**102**	**108**	**114**	**120**	**126**	**666**



In this example there are both floating row totals and floating column totals, and the first two columns are used as the horizontal formatting template.
Resizing to 8 rows and columns gives this:

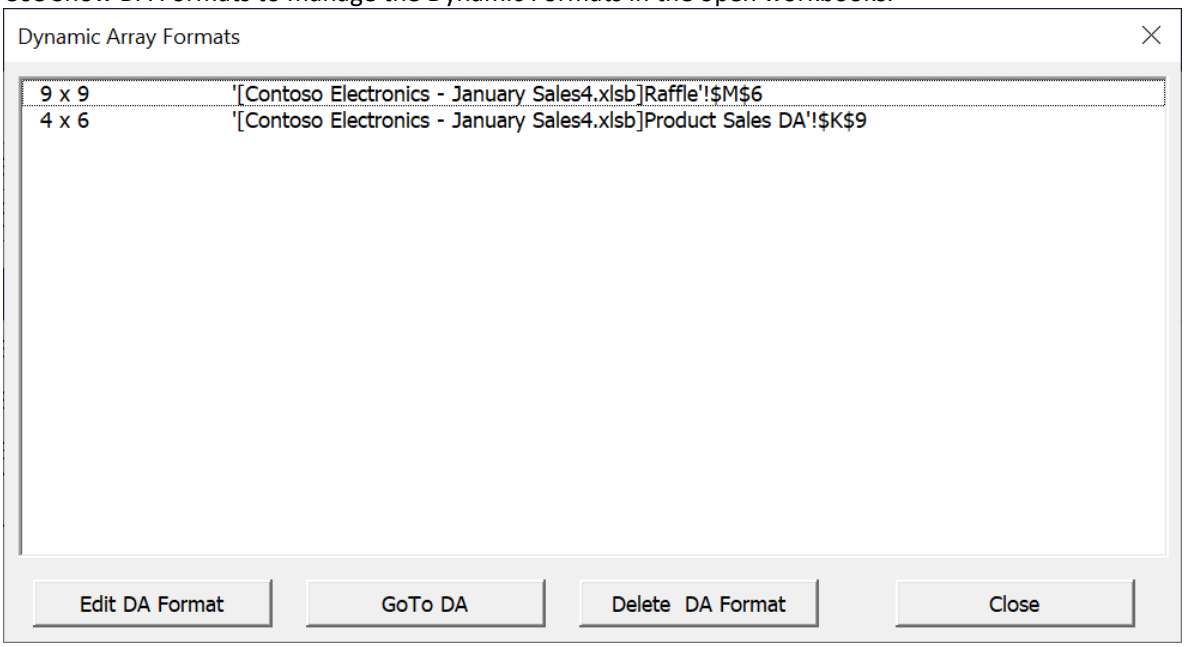
1	2	3	4	5	6	7	8	**36**
9	10	11	12	13	14	15	16	**100**
17	18	19	20	21	22	23	24	**164**
25	26	27	28	29	30	31	32	**228**
33	34	35	36	37	38	39	40	**292**
41	42	43	44	45	46	47	48	**356**
49	50	51	52	53	54	55	56	**420**
57	58	59	60	61	62	63	64	**484**
232	**240**	**248**	**256**	**264**	**272**	**280**	**288**	**2080**

Reformat This DA

Use this button if you want to see the result of applying the chosen formatting template rows and columns.

Show DA Formats

Use Show DA Formats to manage the Dynamic Formats in the open workbooks.



Edit DA Format

Selecting a Dynamic Array Format and clicking Edit DA Formats takes you to the Choose Dynamic Array Format Copy areas.

GoTo DA

Selecting a Dynamic Array Format and clicking GoTo DA activates the top left cell of the dynamic array.

Delete DA Format

Selecting a Dynamic Array Format and clicking Delete DA Format removes the dynamic array formatting metadata from the workbook.
It does not change the current formatting of the dynamic array but reverts to static formatting.

SpeedTools What-If Tables Manager

What If Manager enables you to speed up and Manage Excel What-If Data Tables.

- Slow Excel Data Tables calculate much faster when converted to FastExcel What-If Tables
- Convert Excel What-If Data Tables to FastExcel What-If Data Tables
- Convert FastExcel What-If Data Tables back to Excel What-If Data Tables

FastExcel What If Calculation

Automatic Calculation Mode

Every time Excel calculates all the FastExcel what If Tables recalculate.

Manual Calculation Mode

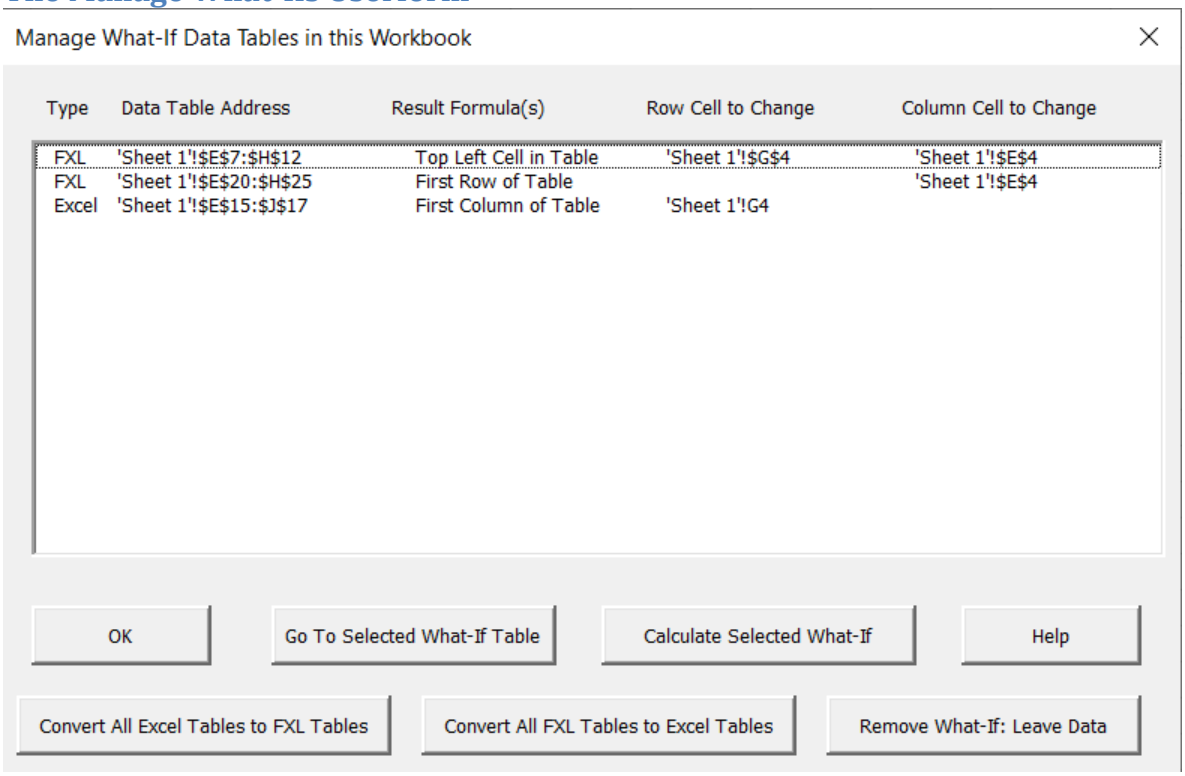
Every time Excel calculates all the FastExcel what If Tables rcalculate.

Partial (Semi-automatic) Calculation Mode

The FastExcel What If Tables only recalculate when one of the FastExcel What-If Manager buttons "Recalc All FXL What-Ifs" and "Recalc the Selected FXL What-If" are clicked.

In Partial calculation mode the FastExcel What-If Tables do NOT recalculate when F9 or Calculate Now are pressed.

The Manage What-Ifs Userform



This example shows the 2 FXL What If Tables and 1 Excel Data Table in the workbook.

Go To

You can double-click an entry or select an entry and click the Go To Selected button to go to its location on its worksheet.

Convert Excel What-Ifs <-> FXL What-Ifs

You can convert all the What-If tables between the Excel and FastExcel versions in either direction.

Calculate Selected What-If

If the active cell is within a FastExcel What-If Table it will be recalculated.

Remove What-If: Leave Data

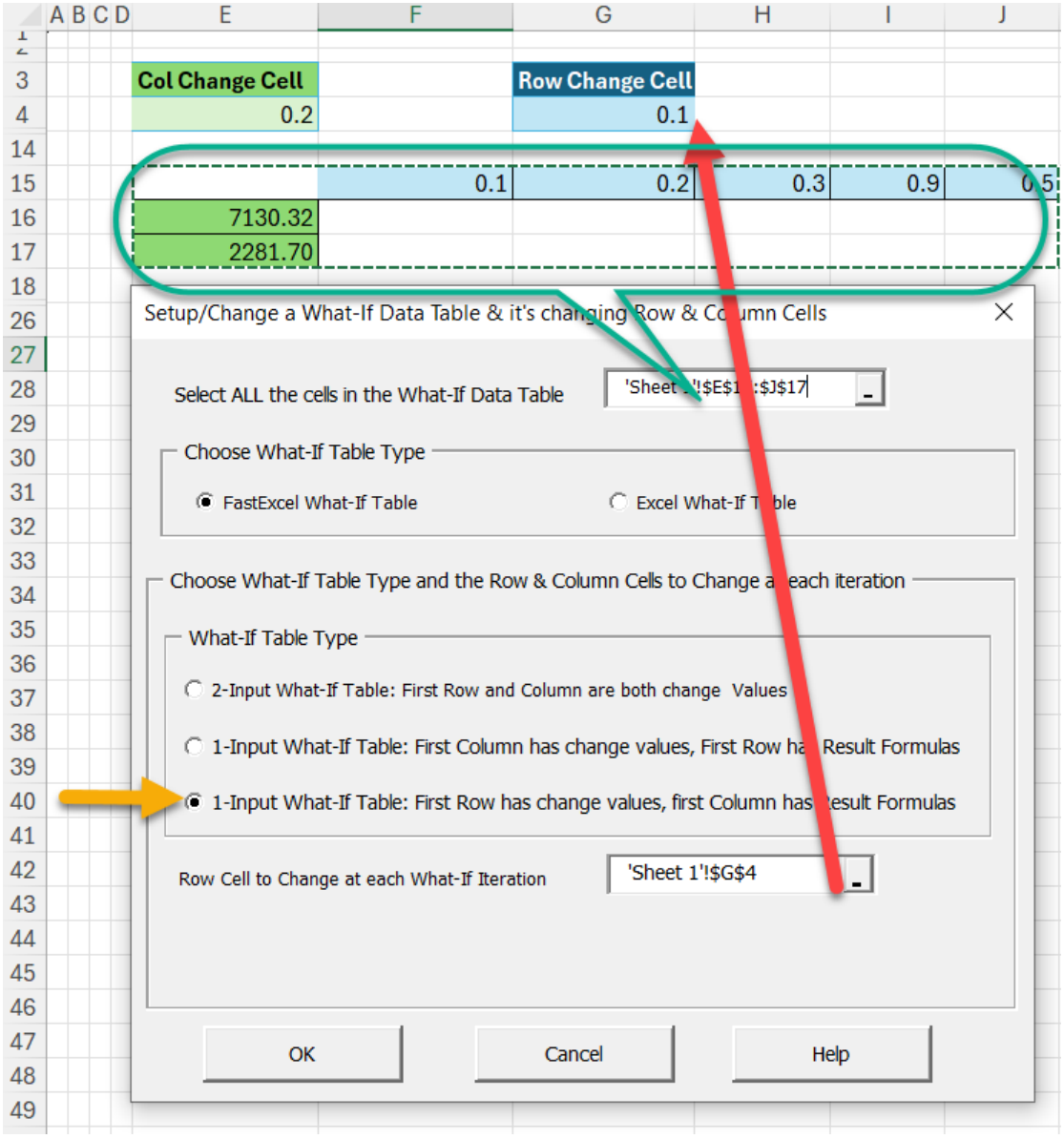
This removes the What-If calculation part from the selected Table. The Result Formulas themselves and the change values are left intact.

Setup/Change a What-If Table

This command provides a simple method of creating either an Excel or FastExcel What-If table from a skeleton on the worksheet: this example skeleton has 5 change values in the first row and 2 result formulas in the first column.

	0.1	0.2	0.3	0.9	0.5
7130.32					
2281.70					

1. **Select all the cells in the skeleton.**
2. **Choose the Table Type.**
3. **Select Cell to change.**


Col Change Cell
0.2
Row Change Cell
0.1
0.1
0.2
0.3
0.9
0.5
7130.32
2281.70
Setup/Change a What-If Data Table & it's changing Row & Column Cells
Select ALL the cells in the What-If Data Table
'Sheet 1'!$E$15:$J$17
Choose What-If Table Type
FastExcel What-If Table
Excel What-If Table
Choose What-If Table Type and the Row & Column Cells to Change at each iteration
What-If Table Type
2-Input What-If Table: First Row and Column are both change Values
1-Input What-If Table: First Column has change values, First Row has Result Formulas
1-Input What-If Table: First Row has change values, first Column has Result Formulas
Row Cell to Change at each What-If Iteration
'Sheet 1'!$G$4
OK
Cancel
Help

What-IF Data Table examples

	E	F	G	H	I	J
3	Col Change Cell		Row Change Cell			
4	0.2		0.1			
5						
6	Result Formula					
7	14118.04	0.1	0.2	0.3		
8	0.1	14565.1	14124.0	13387.3		
9	0.2	14118.0	13690.6	12976.4		
10	0.3	13361.7	12956.5	12282.4		
11	0.9	2782.9	2699.9	2555.9		
12	0.5	11024.7	10691.7	10133.3		
13						
14						
15		0.1	0.2	0.3	0.9	0.5
16	7130.32	7130.3	6914.4	6553.8	1371.3	5403.2
17	2281.70	2281.7	2212.6	2097.2	438.8	1729.0
18						
19						
20		7130.32	2281.70	4563.41		
21	0.1	7356.1	2354.0	4707.9		
22	0.2	7130.3	2281.7	4563.4		
23	0.6	4762.2	1523.9	3047.8		
24	0.9	1405.5	449.8	899.5		
25	0.5	5568.0	1781.8	3563.5		

The example **shows** the 3 types of what-if table:

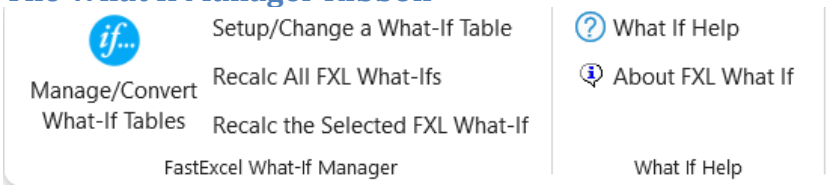
***Two-Input Data Table*: E7:H12 with a 5 x 3 table of calculated results. These can only have one result formula but show every combination of row and column of change values.**

- Result Formula in the top-left cell E7 that is dependent on the Change Cells in E4 and G4
- Column of change values to take at each iteration and put into the the Col Change cell: E4
- Row of change values to put into the Row Change cell: G4

***One-Input Data Tables* by row or column. These can have multiple result formulas but only one set of change values.**

- **Column of Result Formulas** & Row of Change Values: E15:J17.
  - 2 Result Formulas in E16:E17
  - 5 Row change values in F15:J15 to put into cell G4 at each successive iteration.
- **Row of Result Formulas** and Column of Change Values: E20:H25
  - 3 Result formulas in F20:H20
  - 5 Column Change Values in E21:E15 to put into cell E4 at each successive iteration.

The What If Manager Ribbon



Manage/Convert What-If Tables

Launches the What-If Manager showing all the Excel and FastExcel What If Tables in the active workbook.

Setup/Change a What-If Table

Launches a simple wizard for creating or changing an Excel or FastExcel What If Table.

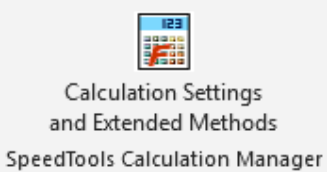
Recalc FXL What-Ifs

Calculates the selected FastExcel What If or all Fastexcel What-Ifs in the active workbook.

About FXL What If

Show the build number of the currently loaded What If Manager together with the latest available build number.

SpeedTools Calculation Manager

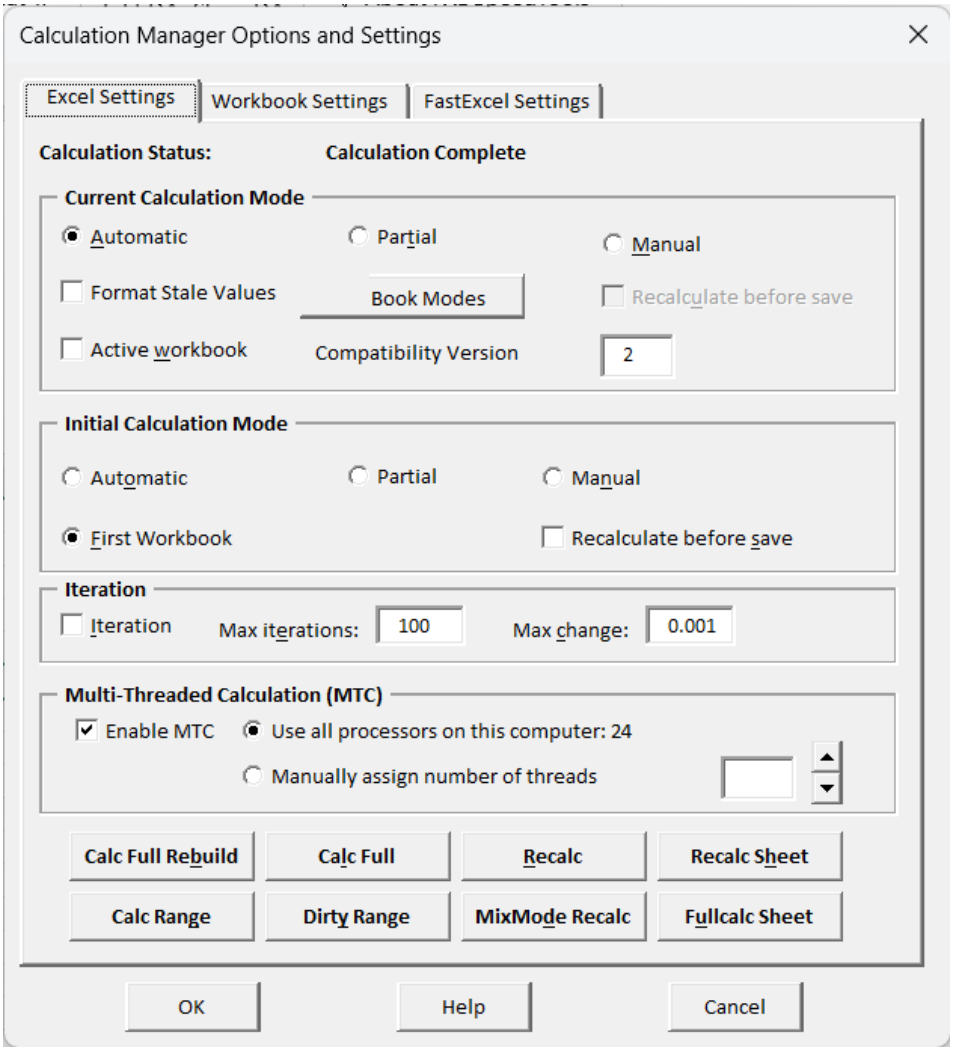


SpeedTools Calculation Manager gives you:

- Extended Calculation Methods only available with FastExcel SpeedTools or FastExcel Manager Pro for Mac.
    - Active workbook mode – only calculate the active workbook
    - Control which worksheets get calculated (MixMode worksheets)
    - Control Initial calculation mode at Excel startup
    - Restore calculation mode after workbook open
    - Calculate MixMode sheets at Workbook open
- Full Control of all Excel's calculation settings
- Call the Calculation methods from buttons on the FastExcel Calculation Settings Form.
    - Calculate Full Rebuild Dependencies
    - Calculate Full all open workbooks
    - Recalculate all open workbooks
    - Recalculate Worksheet
    - Full Calculate Worksheet
    - Calculate Range
    - Mixed Mode Recalculation
- Accurate timing for each calculation method
    - On Windows timing accuracy is at the Microsecond level, although in practice other factors such as Windows multi-tasking prevent timing tasks reaching this level of accuracy.
    - On Mac timing is accurate at best to around 4 milliseconds.

Calculation Manager Options and Settings

The settings are in three groups: Excel Settings, Workbook Settings and FastExcel Settings.



Excel Calculation Settings

Excel's default methods for Determining Calculation Mode.

By default Excel uses the same calculation mode for all open workbooks. The Calculation mode is initially set from the first workbook opened and is not changed when other workbooks are opened. Use the extended FastExcel calculation methods to override Excel's standard calculation methods

Excel uses the same calculation mode for all open workbooks.

Excel sets the initial calculation mode from the first (previously saved) workbook opened. This initial calculation mode will not change when additional workbooks are opened, and will only change when either the user or a VBA program changes it. If you start Excel with a brand new empty workbook and make changes to that workbook then the calculation mode is Automatic.

If a manual mode workbook is opened in automatic mode the workbook will be calculated.

FastExcel gives you many extended calculation methods which you can use to change and control the default Excel calculation settings.

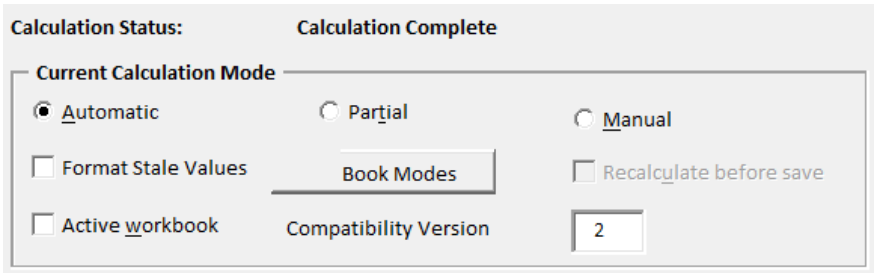
The extended Calculation methods only available with FastExcel include:

- Active workbook mode
- Initial calculation modes
- MixMode worksheets selection and options
- Restore calculation mode after open
- Calculate MixMode sheets on open

These extended calculation methods only work when you have either FastExcel SpeedTools or FastExcel Manager Pro for Mac loaded.

Excel Current Calculation Mode

You can use this form to view and change Excel's current calculation status and settings.



*By default the current Excel calculation mode applies to all the open workbooks. Also **all** the open workbooks are recalculated, not just one*

Excel's default calculation mode is at the Excel session level. This means that Excel will use the same calculation mode for all open workbooks, regardless of the calculation mode that was saved with an individual workbook.

Whenever Excel calculates the default behavior is to calculate all the open workbooks, rather than just the active workbook

Automatic

Formulas are recalculated automatically whenever anything changes so that the workbook(s) are always calculated.

Partial

Similar to automatic except that Excel Data Tables, Python formulas and automatic Pivot Tables are not automatically calculated. This can be useful with large workbooks because Excel Data Tables cause multiple calculations of the workbook.

Manual

The status bar will also show "Calculate" if there are circular references or ForceFullCalculation has been turned on.

Formulas are only recalculated when the user requests it by pressing F9 or the FastExcel recalculate button. When the workbook is not completely calculated the status bar shows "calculate".

Recalculate before save.

If in Manual mode checking this option will cause Excel to recalculate uncalculated formulas in the workbook each time it is saved.

Format Stale Values

When in Manual or Partial calculation mode formulas that are currently uncalculated (Stale Values) will have Stale Value formatting applied.

Compatibility Version

Determines what compatibility version of the calculation engine is currently being used for the Active workbook.

FastExcel Active Workbook Mode

Open multiple workbooks but only calculate the active one.
Set different calculation modes for each open workbook with ***Set Book Modes.***

Whilst this option is active you cannot copy and paste between workbooks.

Note: Active workbook mode is only available with FastExcel SpeedTools and FastExcel Manager Pro for Mac

Excel normally calculates **all** the workbooks you have open at each calculation. This can cause inconvenience and slow calculation when you have multiple workbooks open.

FastExcel allows you to set Active Workbook Mode so that Excel will only calculate the active workbook.

When you set Active Workbook Mode the default is that each open workbook is in Manual calculation mode, so that you have to press F9 to calculate the Active Workbook.

For more complicated situations FastExcel allows you to set different calculation modes for each workbook.

For example suppose one workbook is set to Automatic and another to Manual:

- If the Automatic workbook is active then it will be automatically calculated at any change to the workbook, but the inactive manual workbook would not be calculated.
- If the Manual workbook is active then it will only be calculated when you press F9, and the inactive Automatic workbook will not be calculated.

You can set the calculation mode for each of the open workbooks (and the default mode) using the **Set Book Modes** button. This book calculation mode is saved with each workbook.
If you use active workbook mode with a workbook in Automatic mode then it will be recalculated whenever you make it active.

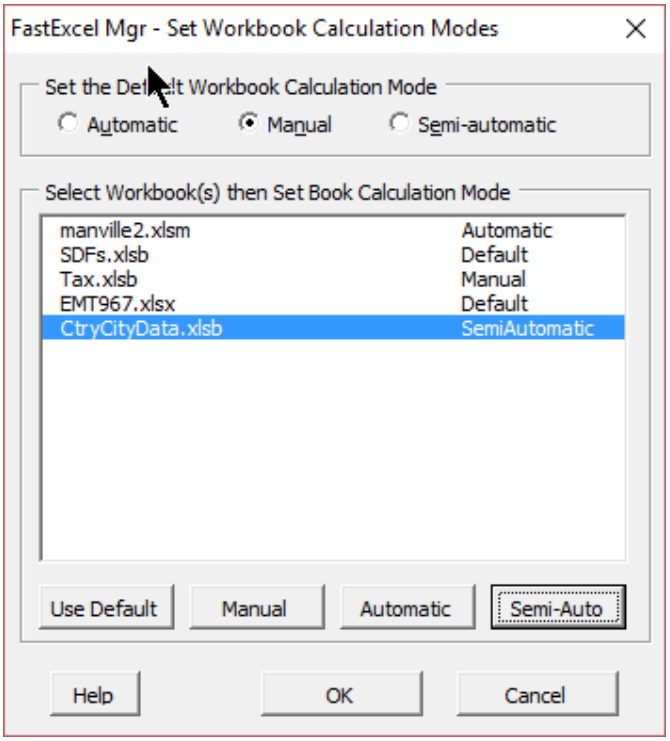
Once you set this option it will still be active the next time you open Excel.

Whilst this option is active you cannot copy and paste between workbooks.

You can use Active Workbook Mode for the workbooks at the same time as Mixed Mode for the worksheets, but Mixed Mode is not dependent on Active Workbook Mode.

Excel Set Book Modes

Note Set Book Modes is only available with FastExcel SpeedTools and FastExcel Manager Pro for Mac



Default Active Workbook Calculation Mode

The Default Workbook calculation Mode only applies when you have selected Active Workbook Mode.

You can change the FastExcel default workbook calculation mode whenever you want. The FastExcel default will apply to all workbooks in Active Workbook Mode until changed, and will be recalled when you next open Excel.

The default workbook calculation mode is used in active workbook mode to calculate a workbook with no assigned workbook calculation mode, or one assigned a mode of default.

The initial default mode is Manual.

Set Book Calculation Modes

The workbook calculation mode for each workbook can be:

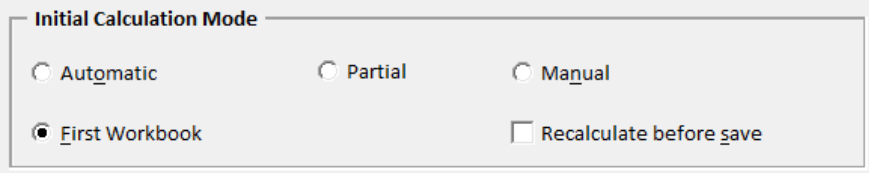
- Manual
- Automatic
- Semi-automatic (Automatic except Tables)
- Default: The workbook will use the current default workbook calculation mode.

To change the workbook calculation mode for one or more workbooks, select the workbooks and press one of the calculation mode buttons.

Excel Calculation Settings: Initial Calculation Mode

Note: Set Initial Calculation Mode is only available with FastExcel SpeedTools and FastExcel Manager Pro for Mac

You can use these settings to control which calculation mode Excel will use when opened.



By default the first workbook opened sets the mode until it is changed by the user

Force Excel to open in Manual mode to prevent your workbook being accidentally recalculated when you open it

By default Excel sets the initial calculation mode from the first workbook opened, and does not automatically change it when another workbook with a different setting is opened.

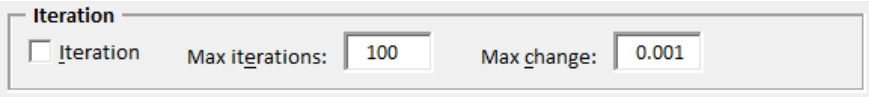
FastExcel allows you to control the mode to be used when Excel first opens. This initial mode can be:

- **First Workbook (Excel default)**
- **Manual**
- **Manual with recalculate before save**
- **Automatic**
- **Partial**

Limitation: When you use FastExcel initial calculation mode the empty workbook (Book1) that Excel creates when it is started will prompt you to save it when you close Excel.

Excel Iteration

These settings control how Excel handles circular references.



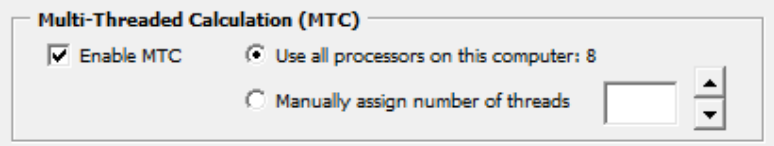
Iteration

When checked Excel will attempt to resolve formulas that are linked by circular references by repeatedly calculating them.

Maximum iterations and Maximum change

Excel will stop the repeated calculation of the circular references as soon as either the maximum number of iterations has been reached or the maximum change in the values of all of the formulas is less that the specified maximum change.

Multi-threaded calculation Settings



Enable MTC

When Checked Excel (Windows Excel 2007 and Mac Excel version 16 and later) will split the calculation process over multiple threads. This will improve calculation speed on many problems, particularly with large workbooks and multiple recalculations.

Use all Processors/Manually Assign threads

You can control the maximum number of threads to be used by Multi-Threaded calculation

Excel Calculation Buttons



The calculation timer for these buttons is controlled by your FastExcel settings

The Excel Calculation Settings Tab also contains buttons to request various types of calculation. **FastExcel** will show the time taken by the calculation if the calculation timer for buttons is switched on.

Using FastExcel defaults Excel will calculate all enabled worksheets in all open workbooks, and will calculate all MixMode sheets with a Full calculate, and selected MixMode sheets with a Sheet calculate.

Calc Full Rebuild (Shift-Ctrl-Alt-F9)

By default MixMode sheets are calculated if Mixed Mode Full Calculate is on

Calculate Full Rebuild rebuilds the dependency trees and then fully calculates all formulas in all open workbooks. This option is only available from Excel 2002 onwards.

Calc Full (Ctrl-Alt-F9)

By default MixMode sheets are calculated if Mixed Mode Full Calculate is on

Calculate Full fully calculates all formulas in all open workbooks.

Recalc (F9)

Recalculation is the default Excel calculation process. Only changed cells and formulas dependent on changed cells are calculated

Recalculate recalculates all formulas in all open workbooks that have been changed or are volatile, or are dependent on a cell that has been changed or recalculated.
By default MixMode sheets are calculated if calculation mode is Manual and Mixed Mode Manual is on.

Recalc Sheet (Shift-F9)

Sheet Calculate re calculates the selected Sheets assuming that all precedent worksheets and linked workbooks have been correctly calculated.

Recalculate Sheet recalculates all formulas on each selected sheet that have been changed or are volatile or are dependent on a cell on the selected sheet that has been changed or recalculated. By default MixMode sheets are calculated if Mixed Mode Selected Sheets is on.

Fullcalc Sheet (Alt-Shift-F9)

Full Calculate Sheet calculates every formula on the selected Sheets assuming that all precedent worksheets and linked workbooks have been correctly calculated.

Full Calculate Sheet calculates all formulas on each selected sheet. By default MixMode sheets are calculated if Mixed Mode Selected Sheets is on.

Calc Range (Alt-F9)

Use FastExcel Calculation Control->FastExcel Settings to choose between the 2 available Range Calculate methods (with or without dependencies internal to the selected range) and the number of Range Calculate trials (default 3). Range Calculate then shows the median time of all the trials

Range Calculate calculates the currently selected range of cells.
If more than one Area is selected; each Area is calculated left-to-right and top-to-bottom, in the sequence that the areas were selected.
If more than one worksheet is selected then only the selected range on the active worksheet is calculated.
If one or more of the selected cells is part of a multi-cell array formula then all the cells in the array formula will be included in the calculation.
If Calculation is set to Automatic both the time taken to calculate the range in Manual mode and the workbook recalculation time are shown.

Dirty Range

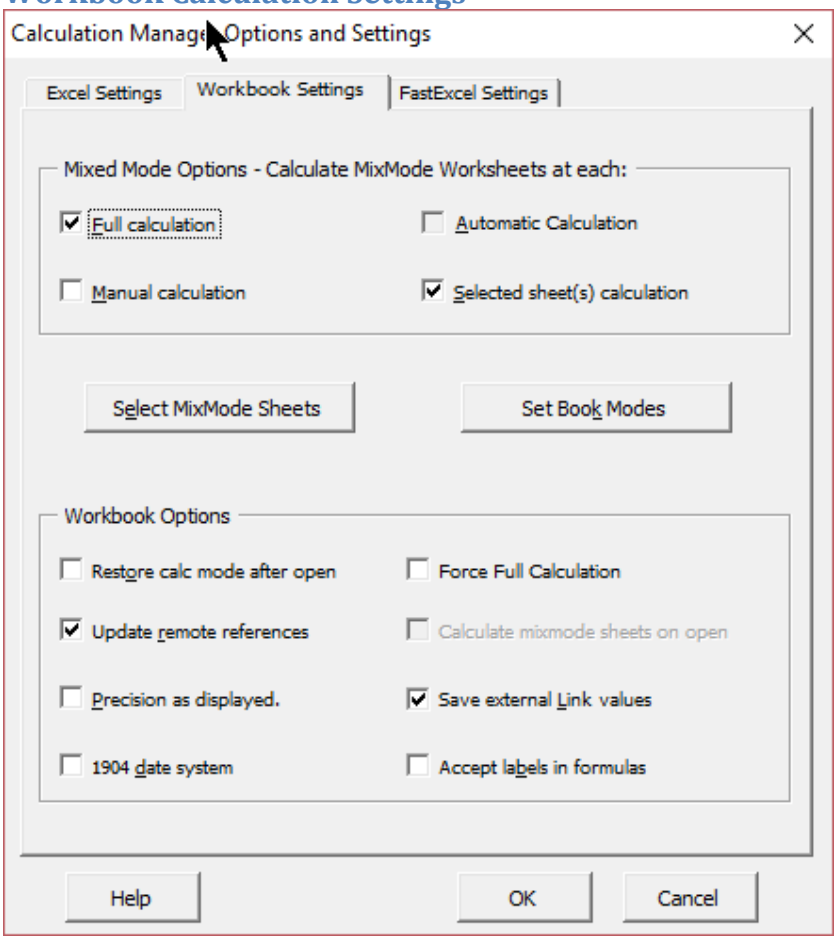
Flags the selected cells as uncalculated. This option is only available in Excel 2002 or later. In automatic mode this will trigger a recalculation.

MixMode Recalc (Ctrl-Shift-F9)

Recalculates both MixMode and ordinary sheets.

MixMode Recalc is similar to Recalc (F9), except that the recalculation also includes all Mixed Mode sheets.

Workbook Calculation Settings



Workbook Calculation Settings: Mixed Mode

Note: FastExcel Mixed Mode is only available with FastExcel SpeedTools and FastExcel Manager Pro for Mac

Options for calculating MixMode Worksheets

Make some sheets calculate automatically and others only when you request calculation

FastExcel allows you to mix calculation modes for different worksheets within a workbook. **The default settings are designed to allow you to use either automatic or manual calculation for normal sheets, but only calculate specified MixMode sheets when you need to**. You can use mixed mode at the same time as Active Workbook mode.

- You can choose which worksheets to designate as MixMode sheets.
- MixMode worksheet settings are stored with the workbook and persist between Excel sessions.
- You can use default settings or customize the settings.
- MixMode worksheets can be combined with the new calculate active workbook only mode and the ability to set calculation mode by workbook.

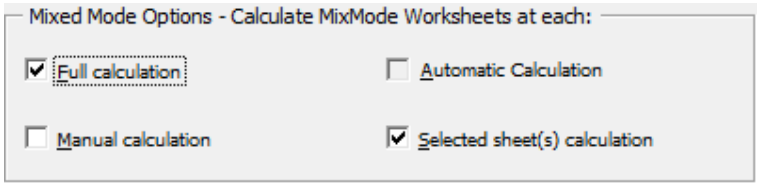
Mixed Mode default settings

If Active Workbook Mode is ON then only the active workbook is calculated: if Active Workbook Mode is OFF then all open workbooks are calculated

When you use FastExcel's default MixMode settings Excel calculates as follows:

- In automatic calculation mode MixMode sheets are **not** recalculated automatically.
- F9 or the Calculate button will **not** recalculate MixMode worksheets.
- Ctrl/Shift/F9 or the **Calculate MixMode Sheets button** will recalculate the workbook(s) **including MixMode** sheets.
- Shift/F9 or the Calculate Sheet(s) button will calculate **all the selected sheets including any selected MixMode sheets**. If in automatic mode the workbook(s) will then also be recalculated.
- Ctrl/Alt/F9 or the Full Calculate Button will calculate all formulas on all sheets in all workbooks, **including MixMode sheets**.

Controlling Mixed Mode Settings



Mixed Mode calculation of MixMode sheets occurs for all open workbooks unless you have selected Active Workbook Mode.

You can control MixMode sheets will be calculated using these options:

- **Full Calculation**: when this option is checked all formulas including those on MixMode sheets in the active workbook will be calculated when you press Ctrl-Alt-F9 or the FastExcel Full Calculate button. The default is checked.
- **Manual Calculation**: when this option is checked pressing F9 or FastExcel's Recalculate button will recalculate formulas that are flagged as uncalculated, including those on MixMode sheets in the active workbook. The default is not checked.
- **Selected Sheet(s) Calculation**: when this option is checked pressing Shift-F9 or FastExcel's Calculate Sheets button will recalculate formulas that are flagged as uncalculated on all selected sheets, MixMode or enabled. The default is checked.
- **Automatic Calculation**: this option is permanently disabled: Excel's automatic calculations do not recalculate MixMode sheets (except when you enable them of course).

You can select which sheets will be MixMode sheets using the Select MixMode Sheets button.

When a workbook is opened

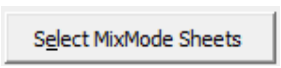
If Excel is in automatic mode all MixMode sheets are recalculated.

If Excel is in manual mode any MixMode sheets are not recalculated unless the "Calculate MixMode sheets on open" option (Workbook Calculation Settings tab) was selected when the workbook was saved.

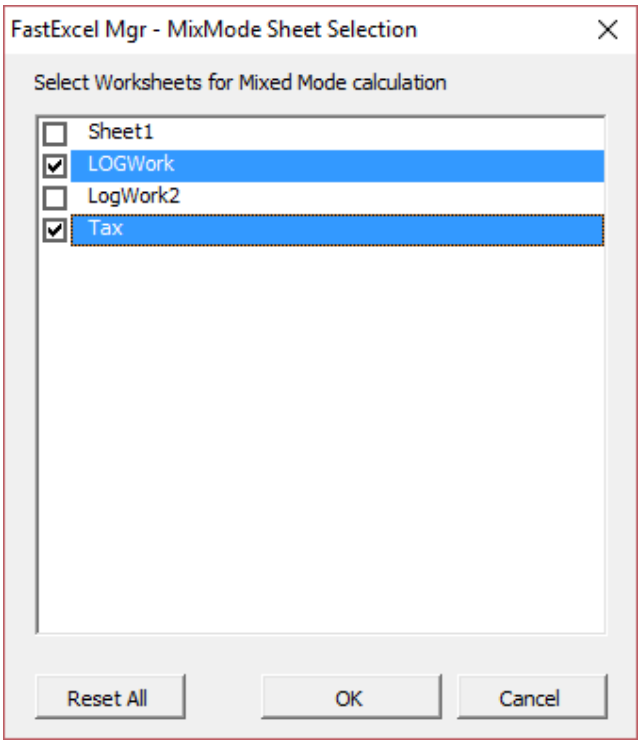
Using Range Calculate in Mixed Mode

Range calculate or Alt-F9 always calculates the selected range even on a MixMode sheet.

Workbook Calculation settings: Choosing the MixMode Sheets



You can choose which sheets to process in Mixed mode using the **Select Mixmode Sheets** button on the Active Workbook Calculation settings tab. FastExcel stores these settings for each worksheet in the workbook, and restores the last saved settings when the workbook is opened.



Use this command to select the sheets you want to be handled as MixMode sheets. These sheets will be calculated according to the options set for Mixed Mode calculations.

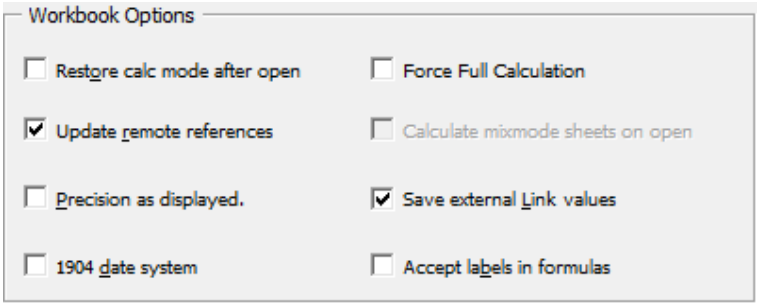
You can also use this tool to explore the possible effect of combining sheets.

In cases where there is a significant workbook calculation overhead you may be able to simulate the effect of combining the sheets causing the **bottleneck** by setting suspect worksheets to MixMode and monitoring the effect on workbook recalculation time.

NOTE: When using FastExcel SpeedTools or FastExcel Manager Pro for Mac these settings are saved when you save a workbook and restored when you open a workbook. These settings use Excel's Worksheet Enable Calculation property.

This behavior is different to the standard Excel behavior.

Workbook Calculation Settings: Workbook Options



Restore Workbook Calculation Mode after open

This option only functions when:
-the workbook has been saved
-and the workbook is subsequently opened.

When Restore calculation mode after open is checked FastExcel will automatically reset the calculation mode **after the workbook has been opened** to the workbook calculation mode. You can set the workbook calculation mode using Set Book Modes, or let it default to the default workbook calculation mode, which you can also set in Set Book Modes.

Note that using this option does NOT prevent Excel calculating a workbook when it is opened and Excel is in Automatic mode.

Use "Restore calculation mode after open" instead of Active Workbook Mode if you want to force Excel into the calculation mode of this workbook whenever it is opened, but do not want to switch calculation mode whenever a workbook is activated.

Force Full Calculation

This option is only available in Excel 2007or later and Mac Excel 365.

If checked every calculation on this workbook will be a full calculation rather than a recalculation, and the time taken to rebuild the dependency trees will not be needed at workbook open time or when the workbook is edited.

Switch on Force Full Calculation if adding a row at the top of a worksheet is very slow and Full Calculate time is close to Recalculate time.

Calculate MixMode sheets on open

Only works if FastExcel was active when the workbook was saved	Calculate all worksheets including MixMode when the workbook is opened in manual mode. In Automatic mode MixMode sheets are always recalculated on open, so you cannot select this option when calculation mode is automatic.

Update Remote References

If TRUE automatically update any remote references (DDE Links to other programs) whenever Excel recalculates.

Save External Link Values

If TRUE Excel will save values for the links to external workbooks. I recommend keeping this option as TRUE

Precision as Displayed

Checking this box will force Excel to calculate to the number of decimal places that appear as a result of your formatting, ***and will permanently change any numbers stored in cells***.
You need to be sure you have thought through the full implications of this before using it.
Precision as Displayed slows down calculation.

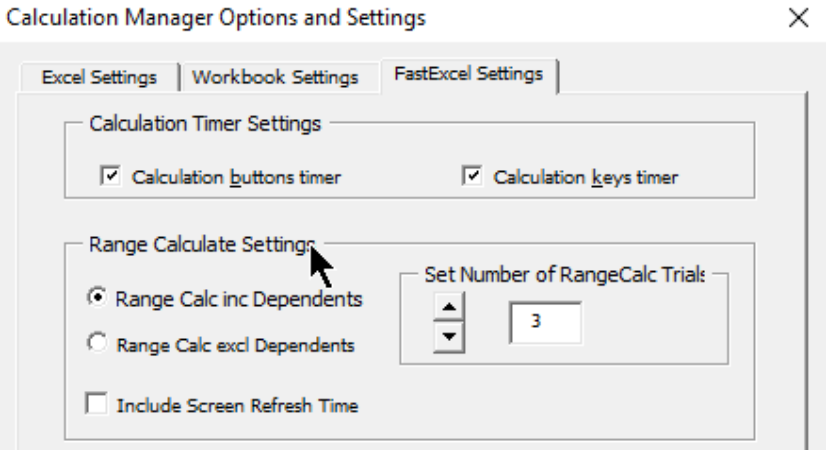
1904 Date System

If TRUE changes the starting date from which all dates are calculated from January 1 1900 to January 2 1904.

Accept Labels in Formulas

This allows Excel to use natural language labels in formulas. Because there are circumstances when this will give you unexpected or ambiguous results, ***I recommend you do NOT use this feature.***

FastExcel Settings

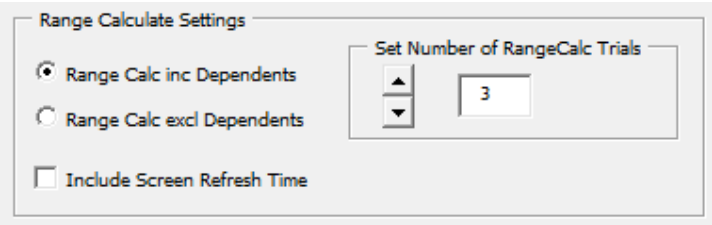


Calculation Timers for buttons and keys

You can control the FastExcel calculation time display for the calculate buttons.

When checked you will see a message box after the calculation showing the time taken by the calculation.

Range Calculate Settings



Range Calc including or excluding Dependents

In Excel 2007 and later there are 2 different Range calculate methods. One includes within-range dependency calculations and the other ignores within-range dependencies and calculates row by row strictly left-to-right and top-to-bottom. Range Calculate always ignores dependencies outside the selected range, although the selected range will be expanded to include all the cells in multi-cell array formulas.

Set Number of RangeCalc Trials

Because RangeCalc timing can be sensitive to Windows multi-tasking it is better to allow a number of trials. The Median (middle-most) calculation time will automatically be used, thus discarding incorrectly high and low values.

You can set the number of trials: the default number is 3.

Include Screen Refresh Time

If checked timings for Range Calculate will include the time used by Excel to refresh the visible window.

Getting Consistent Results from FastExcel V5 Timing

Sometimes you will find that your **FastExcel** timing results are not repeatable, even though **FastExcel** measures elapsed time using a high-resolution timer with microsecond accuracy.

Power Saving and Multi- Core processors:

Power saving features, particularly on dual-core processors, may dynamically vary the speed of your processor. This can create problems for consistent timings using the high-resolution timer. If timing consistency is important try switching off the power-saving features of your processor.

Excel minimizes the number of calculations:

When Excel calculates it remembers:

- Which cells have been calculated.
- The optimum calculation path used for the final answer.

The second calculate is often much faster than the first

So the next time Excel calculates it can save time by:

- Only calculating the cells that have changed (or depend on cells that have changed)
- Re-using the last calculation path

Why FastExcel Timing results may vary from run to run:

Both Excel and Windows cache information to speed up subsequent operations

Both Excel and Windows try to optimize performance by storing recent information in memory. This includes the calculation path used by the most recent calculation.

So Excel's second calculate is often significantly faster than the first.

Windows is a multitasking system so Excel does not get the whole system to itself

Excel also saves some of this information, including the most recent calculation path, with the workbook.

Excel's background calculation process can easily be interrupted

Windows is a multitasking, paged virtual memory operating system so other system tasks are always running at the same time as Excel. Try and make sure there are no other user tasks active whilst you are using **FastExcel.**

Excel's background calculation process can be interrupted if you press any key or click the mouse whilst **FastExcel** is active. This may give you inaccurate timings.

FastExcel V5 can help you quickly find out how much memory Excel is using.

Calling SpeedTools functions from VBA

The easiest method of calling the SpeedTools functions from VBA is to use Application.Run.

It will convert your function parameters to an appropriate type as required.
The only major drawback to Application.Run is that the parameters are handled By Value as opposed to By Reference, which means that each parameter is copied before being passed to the function. This is fine for scalar values and objects such as a Range object but is slow for arrays (objects get passed as pointers so the function has to know how to handle whatever data structure or object the pointer points to).

You can pass Application.Run either the name of the function/sub as a string or as a Register ID if it is an XLL function. And you can use Evaluate to convert the name of the XLL function to its Register ID, by calling Evaluate("FunctionName") rather than Evaluate("FunctionName()").

If you are repeatedly calling the function using Application.Run it is much faster to use the Register ID rather than the function name.

```
Sub TestRun()
Dim var As Variant
Dim rng As Range
Dim dArray(1 To 10) As Double
Dim j As Long

''' using a string
var = Application.Run("Reverse.Text", "Charles")

Set rng = ActiveSheet.Range("B9:B20") ''' using a range
var = Application.Run("Reverse.array", rng)

''' using a double array
For j = 1 To 10
dArray(j) = j
Next j
var = Application.Run("Reverse.array", dArray)

''' multiple parameter types
Dim str1 As Variant
str1 = Application.Run("Rgx.Substitute", "123456ABC", "[0-9]", "Z", 0, False)

MsgBox str1(1)  ''' Returns a 1-dimensional variant

''' using a register ID for the function: faster if large number of calls to the function
Dim jFunc As Long
jFunc = Evaluate("Rgx.Substitute")
str1 = Application.Run(jFunc, "123456ABC", "[0-9]", "Z", 0, False)
MsgBox str1(1)

End Sub
```

Using FastExcel calculation methods from VBA

If you are using any of FastExcel's additional calculation modes and options and your VBA program is requesting Excel calculations you should use FastExcel's enhanced calculation methods.

If you use Excel's standard calculation methods either FastExcel's additional calculation modes and options will not operate or you may get unexpected results.

You can access FastExcel's calculation methods from VBA either by using Application.Run or by setting a reference in your VBA project to SpeedToolsV4: Alt-F11 to go to the VBE, then Tools→References and check the box beside SpeedToolsV4.

The available calculation methods are:

SpeedToolsV4.FxlRangeCalc

FxlRangeCalc calculates the selected range on the active sheet in the same way as the FastExcel Calculate Range button.

SpeedToolsV4.FxlSheetCalc

FxlSheetCalc recalculates the selected worksheet(s) in the same way as the FastExcel Recalculate Sheets button.

SpeedToolsV4.FxlFullSheetCalc

FxlFullSheetCalc full calculates the selected worksheet(s) in the same way as the FastExcel Full Calculate Sheets button.

SpeedToolsV4.FxlBookCalc

FxlBookCalc recalculates workbooks in the same way as the FastExcel Calculate Book button.

SpeedToolsV4.FxlFullbookCalc

FxlFullBookCalc does a full workbook calculation in the same way as the FastExcel Full Calculate button.

SpeedToolsV4.FxlDisabledSheetsCalc

FxlDisabledSheetsCalc recalculates workbooks including Mixed mode worksheets.

SpeedToolsV4.FxlFullDependCalc

The workbooks will have their dependency trees rebuilt and a full calculation will be done.

MICROTIMER function

The MICROTIMER function is a wrapper function for the Windows high-resolution timer API.

The function returns a double containing seconds. The resolution of the timer is dependent on the clock speed of your PC, but is about 1 microsecond on a 1200MHZ AMD. The function itself takes about 5 microseconds to execute when called from VBA on the same machine.

The function is not volatile, and is primarily designed to be called from VBA Subs/Functions rather than from a worksheet.

MICROTIMER Syntax

MICROTIMER()

MICROTIMER Remarks

To call the function from VBA you must first make a reference from the VBE to FastExcelV5:

- Open the VBE (Alt F11)
- Open a module within your VBA project
- Tools->References and check FastExcelV5.

It is also possible to evaluate the function from VBA as a worksheet function using Evaluate, but this incurs significant extra overhead.

MILLITIMER function

The MILLITIMER function is a wrapper function for the Windows medium-resolution timer API.

The function returns a long containing milliseconds. The resolution of the timer is about 5 milliseconds. The function itself takes about 1 millisecond to execute when called from VBA on the same machine.

The function is not volatile, and is primarily designed to be called from VBA Subs/Functions rather than from a worksheet.

MILLITIMER Syntax

MILLITIMER()

MILLITIMER Remarks

To call the function from VBA you must first make a reference from the VBE to FastExcelV5:

- Open the VBE
- Open a module within your VBA project
- Tools->References and check FastExcelV5.

It is also possible to evaluate the function from VBA as a worksheet function using Evaluate, but this incurs significant extra overhead.

STRCOLID function

The STRCOLID function returns an alphabetic column name (A to XFD) given a column number.

STRCOLID Syntax

STRCOLID(jColNo)

JColNo

jColNo is the number of the column you want to convert to a column name.

STRCOLID Remarks

If the column number is greater than 16384 STRCOLID returns #N/A

STRCOLID can be used either as a worksheet function or from VBA.

The main use of STRCOLID as a worksheet function is to calculate a cell address as a string for use with INDIRECT.