
FastExcel V2 Guide to

Optimising Excel Calculations and Memory

By Charles Williams, Decision Models Ltd



Copyright

Copyright © 2001-2006 Decision Models Limited. All rights reserved.

The information contained herein is supplied without representation or warranty of any kind, is subject to change without notice and does not represent a commitment on the part of Decision Models Ltd.

Decision Models Ltd, therefore, assumes no responsibility and shall have no liability, consequential or otherwise, of any kind arising from this material or any part thereof, or any supplementary materials subsequently issued by Decision Models Ltd.

Decision Models Ltd has made every effort to ensure the accuracy of this material. If you have any questions or comments, contact:

Support@DecisionModels.COM

Document Version

Version is number 2.462 dated 28 February 2006.

Microsoft ® is a registered trademark of Microsoft Corporation

Contents

Optimising Excel Calculation Speed 1

Why is Calculation Speed Important?	1
The Effects of Slow Response Time	1
Faster Excel Calculations.....	2
Comments on Microsoft's Advice	3
<i>MS Says</i> : Build Vertically: Many rows and few columns are better than many columns and few rows.	3
<i>MS Says</i> : Keep raw data on a separate sheet.	3
<i>MS Says</i> : Precalculate Constants.	4
<i>MS Says</i> : Activate the Automatic Except Tables Calculation Option.	4
<i>MS Says</i> : When you only need a few cells to be recalculated replace the = signs in the formulae.....	4
<i>MS says</i> : Where possible a formula should refer only to the cells above/before it:5	
Bottlenecks	6
System Bottlenecks:	6
Opening/Saving Bottlenecks.	8
Excel Calculation Bottlenecks.....	11
Size and Memory Problems.....	14
Optimising Tips and Tricks	16
Prioritise the Calculation Bottlenecks	16
Count the number of cells references and calculations require	16
Look for repeated/duplicated expressions	16
Volatile Functions	16
Lookup Options.....	16
Speeding up Lookup's.....	18
Dynamic Ranges.....	22
Generating Workbooks.....	23
Build Time versus Run-Time:	23
Totals and Conditional Sums.....	24
Subtotals.....	25
Multi-Level and Repeated Calculations:	26
Array Formulae	27
Example: SUM with Multiple Conditions.....	30
Links.....	32

Excel Calculation Information 33

Excel's Smart Recalculation Engine.....	33
Circumstances causing a formula to be evaluated.	34
Circumstances causing a name to be evaluated.	34
Excel's Calculation Process.....	35
Dependencies.....	35
Forward References.....	36
Cell Calculation Sequence.....	36
Workbook and Worksheet calculation sequence.	38
Dependency Trees	39
Within Sheet Dependency Trees:	39
Inter Sheet Dependency Trees:.....	40
Inter Workbook Dependencies	40

Names	40
Controlling Calculation.....	41
Status Bar Shows "CALCULATE"	44
Dependency Tree Limits	44
Full Calculation and Re-Calculation.....	45
Sheet Calculation, Range Calculation, Formula and Part Formula Calculation.	45
Recalculate Selected Worksheet(s) (Shift-F9).....	45
Calculate Selected Range (using Range Calculate in VBA)	46
Calculating a Single Formula or Part Formula	47
Step-by Step Formula Evaluation in Excel 2002.....	47
Controlling Calculation from VBA.	48
Calculation Methods.....	48
Excel 2002 Only:.....	51
Volatile Functions and Actions.....	53
Excel's Volatile Functions.	53
Volatile Actions.....	53
User-Defined Functions.....	55
UDF action being ignored.	55
UDF not recalculating or always recalculating or calculating in an unexpected sequence:	56
Unexpectedly returning #Value or being recalculated more than once.	58
UDF Performance.....	61
Function Wizard	66
Repetitive Calculation Features and Add-Ins.	67
Circular References	67
Intentional Circular References	68
Circular References and Calculation Speed.....	68

Excel Memory 69

Windows Memory and RAM	69
How much RAM do you need?	69
How large a Swapfile do you need?	70
Measuring Memory Used by Excel	71
Measuring System Memory	71
Out of Memory, Memory Limits	72
The published Memory limits.....	73
Workbook Memory limits:	74
Workbook or Excel will not open, and gives "Out of Memory" message.	75
Links, Graphics, Zoom, Charts, and Fonts.	75
Miscellaneous Memory Limits	76
Reducing memory Used	77
Memory Leaks.....	78

The Excel/VBA Community. 79

Acknowledgements:	79
-------------------------	----

Glossary of Terms 80

Index 84

Optimising Excel Calculation Speed

Why is Calculation Speed Important?

Poor Calculation Speed impacts productivity and increases user errors.

Improving calculation speed increases productivity.

User errors, and annoyance level, start to increase, particularly for repetitive tasks.

Users generally refuse to wait longer than 10 seconds.

The Effects of Slow Response Time

Research studies show that a user's productivity and ability to focus on the task deteriorate as response time lengthens.

- **Instant Response:**

For response times of less than about a tenth of a second, users feel that the system is responding instantaneously.

You will be able to use Automatic Calculation mode even when entering data.

- **Sub-Second Response:**

For response times greater than a tenth of a second but still less than about 1 second, users can successfully keep a train of thought going, although they will notice the response time delay. IBM studies from the 1970s and 1980s showed significant productivity gains for users when response times were less than a second.

You will probably need to switch to Manual Calculation mode when entering data.

- **Response time greater than 1 second but less than 10 seconds:**

The user has difficulty in retaining a train of thought, but will probably not have switched to doing a different task whilst they are waiting.

- **Response time greater than 10 seconds:**

Users tend to switch to other tasks.

Faster Excel Calculations

Calculation bottlenecks are the major reason for slow spreadsheets.

Measure calculation time and prioritise the bottlenecks.

Use Decision Models Tips and Techniques to help you eliminate bottlenecks.

Measure and test each change.

*Use Decision Model's **FastExcel** to help you find and prioritise bottlenecks.*

Microsoft Excel can calculate millions of formulae per second.

And each new version of Excel calculates most spreadsheets faster than the previous one.

Usually its possible to speed up the calculation of slow spreadsheets, often significantly:

- Most spreadsheets contain a small number of problem areas, or **calculation bottlenecks**.
- Because Excel is such a flexible spreadsheet system there are many different formulae that can produce the answer you want.
- Some of these formulae are much faster than others.
- In a large spreadsheet it is not always easy to find and prioritise the bottlenecks.

To Speed Up Your Spreadsheet:

- Time the calculation and find the calculation bottlenecks.
- Try a different formula/method that gets the same results.
- Time the calculation again.

This document provides **Tips and Techniques** you can apply to common Bottlenecks, and tells you many little-known **secrets** about Excel's calculation methods and options.

These tips, techniques and secrets have been developed by Decision Models from experience and tests over a number of years.

Because **each Excel Spreadsheet is unique** you should:

- **Measure** the change in calculation time for each change you make.
- **Test** to make sure you are still getting the correct results after a change.

Decision Model's FastExcel is designed to help you:

- Find and highlight bottlenecks
- Optimise the Worksheet calculation sequence
- Time calculation of Workbooks, Worksheets and formula blocks
- Get built-in help for calculation bottlenecks tips and techniques
- Document calculation timings and memory usage in a consistent way

Comments on Microsoft's Advice

This article mainly applies to versions of Excel that are no longer much used.

The Microsoft Knowledge Base has an article called:

[Q72622 –XL: Optimizing Worksheets for Fastest Calculation](#)

I tested the advice given in the article using Excel 97 and Windows 98 on a 300 MHZ Pentium 2 with 128MB RAM. Similar results were obtained using Excel 2000 and Windows 2000.

The results and comments are as follows:

MS Says: Build Vertically: Many rows and few columns are better than many columns and few rows.

Using many columns does not make much difference.

This is no longer true. There is a slight advantage to using as many columns as possible:

Horizontal Worksheet:

262143 formulae in 256 columns and 1024 rows

- Filesize: 8190K
- Memory: 12000K
- Calculate: 430 milliseconds @ 300MHZ

Vertical Worksheet:

262143 formulae in 4 columns and 65536 rows

- Filesize: 9548K
- Memory: 11900K
- Calculate: 580 milliseconds @ 300MHZ

MS Says: Keep raw data on a separate sheet.

Putting your data in a separate sheet is a good design rule, but slows down calculation.

This is excellent design advice but poor calculation speed advice: in fact this usually slows down the calculations, and increases memory usage.

Referencing Data on the Same Worksheet:

768000 formulae referencing 768000 data cells on the same Worksheet.

- Filesize: 32680K
- Memory: 39090K
- Calculate: 1.3 seconds @ 300MHZ

Referencing Data on a Different Worksheet:

768000 formulae referencing 768000 data cells on a different Worksheet.

- Filesize: 32680K
- Memory: 57061K
- Calculate: 4.4 seconds @ 300MHZ

Good design has simple, easy-to-understand formulae, and all constants stored in separate named cells.

Simpler formulae usually calculate faster than complex formulae.

MS Says: Precalculate Constants.

Formulae should be as simple as possible: calculate constants before entering them into a formula.

This is good advice as far as it goes from a performance viewpoint, but a good spreadsheet design rule is that formulae should not contain ANY constants: they should be held in separate cells for audit and maintenance reasons.

The best advice is to look carefully at the formulae that are used most often and consume most time to see if they can be simplified, or split into two formulae one of which is executed less frequently.

MS Says: Activate the Automatic Except Tables Calculation Option.

This is one of Excel's calculation options.

If you are using Excel's little-used table feature this may be sensible.

MS Says: When you only need a few cells to be recalculated replace the = signs in the formulae.

This technique is sometimes useful, but make sure that your workbook is correctly calculated.

Replacing the = sign in a formula makes Excel:

- Update the dependency tree for this formula.
- Evaluate this formula.

You can use this method to check if you think Excel has failed to calculate a formula, or has an incorrect dependency tree.

In Manual mode this method calculates only the formulae with replaced = signs, and does not recalculate other formulae dependent on the results of this formula.

The first time you enter a formula Excel also evaluates it as above.

A natural flow of calculation left-to-right and top-to-bottom is excellent design advice, BUT usually does not affect calculation time.

MS says: Where possible a formula should refer only to the cells above/before it:

Avoid Forward References.

This is no longer true, at least for simple references, from a calculation time perspective.

It's still sensible advice because it helps ease of understanding.

Also extreme forward referencing can slow down data entry.

The forward referencing tests were done using Excel97 and Excel2000 on a single sheet workbook with a continuous chain of formulae, where each formula refers to the cell in before it (Backwards), or after it (Forwards).

Forwards has a chain of formulae running from IV3000 to A1.

Backwards has a chain of formulae running from A1 to IV3000.

Forward Referencing:

768000 formulae in 256 columns and 3000 rows

- Filesize: 23786K
- Memory: 33530K
- Calculate: 1.25 seconds @ 300MHZ

Backwards Referencing:

768000 formulae in 256 columns and 3000 rows

- Filesize: 23975K
- Memory: 33762K
- Calculate: 1.25 seconds @ 300MHZ

There is no significant difference in calculation speed when forward referencing on the same Worksheet, although editing can become slow.

Forward worksheet cross-references can be very slow.

See Using [FastExcel](#): optimise worksheet sequence.

In extreme cases forward referencing can slow down Data Entry.

Slow Data Entry with Extreme Forward Referencing:

Forward referencing on the same Worksheet **will** slow down Excel's rebuilding of the dependency tree. This is not done when calculating, but when the formula is entered or changed. This behavior changed with Excel97.

The effect is only noticeable with very deeply nested forward dependencies on large Worksheets. If you are in Manual Calculate mode and notice a pause when you enter a formula this may be the reason.

In the **Forwards** example above changing the number in IV3000 and tabbing to the previous cell whilst in Manual Calculation mode takes 58 seconds @ 300MHZ.

The equivalent for **Backwards** is nearly instantaneous.

Bottlenecks

Bottlenecks that slow down calculation can occur in:

- Your system
- Excel calculations
- Excel memory and Workbook size
- Excel Settings

System Bottlenecks:

Some hardware and system software features and settings can slow down Excel.

Temporary Files

Temporary files may accumulate in your \Windows\Temp directory (Win95/98/ME), or your \Documents and Settings\local settings\temp directory (Win2K/WinXP).

They are created by Excel for the workbook, and in particular for controls being used by open workbooks. If Excel crashes for any reason these files may not be deleted.

They may also be created by software installation programs.

Too many temporary files can cause problems, so clean them out from time to time, but be careful not to do this if you have done a software install that requires you to reboot your PC, but have not yet done so. In this case you should reboot before deleting the temp files.

An easy way to access your temp directory is from the Windows Start button: Start-->Run-->%temp%-->Ok.

The FastExcel Version 2 Clean Workbook command allows you to clean up your temp directory without leaving Excel.

RAM:

256 or 512 MB of RAM usually works well with Windows XP.

Paging to your virtual memory swap-file is extremely slow. You need enough Physical RAM for the operating system, Excel and your Workbook(s), and today RAM is very inexpensive.

If you have more than very occasional hard disk activity during calculation, you need more RAM.

For Windows 95/98/ME

- 32 MB RAM is fine for small Workbooks.
- 64 MB RAM is OK unless your workbook is larger than about 32 MB.
- 128 MB RAM is excellent except for workbooks containing very large amounts of data.
- You will continue to see small speed improvements with 256MB when using large workbooks.

For Windows ME/NT/2000 it's probably a good idea to have an additional 32 or 64MB, and for Windows XP an additional 128MB.

Excel 2002 (Office XP) can make effective use of more RAM than previous versions. With very large workbooks you may see speed improvements using 384MB or 512MB of RAM.

Excel 2003 has substantially increased memory capacity and can use 1024MB or more of RAM.

If you are using large spreadsheets get the fastest machine you can afford.

Outlook Journalling is often cited as a cause of slow response.

MHZ:

Although it is often possible to improve Excel calculation time by a large factor, as you would expect, Excel always runs faster on a fast machine, so if you have a slow machine you should upgrade to a faster one.

Of course **FastExcel** might give you an alternative solution!

Outlook Journalling

If you have Microsoft Outlook installed make sure that you have Journalling turned off, otherwise your machine may behave like a snail. To turn Journalling off try one of these (depends on your version of Outlook):

- Outlook-->Tools-->Options-->Journal-->Uncheck Excel.
- Outlook-->Tools-->Options-->Preferences-->Journal Options

FindFast (Off97 & 2000) & Fast Search (Office XP)

FindFast has a habit of springing to life whenever you don't need it, and in my opinion it doesn't do anything very useful anyway besides chew up a lot of hard disk space. My advice is to turn it off:

Office 97 & 2000: Use the Control Panel, FindFast Icon and delete the indexes etc.

Office XP: File -->Search-->Other Search Options --> Fast Searching is Enabled --> Search Options --> Advanced --Delete. (If Fast Searching is Enabled does not appear, it is not installed).

Fragmented Swap-File

Make sure your Windows swap file is located on a disk with a lot of space, and that you de-fragment the disk from time to time.

Opening/Saving Bottlenecks.

If one or more of your workbooks seem to open and close more slowly than you think they should, check out these possible problems.

Too many Temp Files.

Try deleting ALL the files and subfolders in your \Windows\Temp directory (Win95/98/ME), or your \Documents and Settings\<user>\local settings\temp directory (Win2K/WinXP). These can build up over time and cause Excel to open files very slowly.

Make sure you have completed any software installations that require a reboot before you delete all the temp files!

Workbook with password-protected structure.

A workbook that has its structure protected with a password (Tools-->Protection-->Protect Workbook) will open and close much slower than one protected without a password.

Used Range problems

Excess used range can cause slow opening, especially if caused by hidden rows/columns with non-standard height/width. Use **FastExcel's Clean Workbook** command.

Outlook Journalling.

If this is turned on it will slow down opening and closing workbooks.

FindFast (Off 97 & 2000) & Fast Search (Office XP)

Make sure you have FindFast and FastSearch turned off.

Office 97 & 2000: Use the Control Panel, FindFast Icon and delete the indexes etc.

Office XP: File --> Search --> Other Search Options --> Fast Searching is Enabled --> Search Options --> Advanced --Delete. (If Fast Searching is Enabled does not appear, it is not installed).

Large number of Controls on Worksheets.

A large number of controls (checkboxes, hyperlinks etc) on worksheets can slow down opening a workbook because of the number of temporary files used. This may also cause problems opening/saving a workbook on a WAN (or even a LAN).

These temporary files may also build up in your windows temporary directory:

- Try deleting all files and subfolders in Windows\Temp or documents and settings\<user>\local settings\temp

Large number of Links to other workbooks.

If possible open the workbook(s) you are linking to before the workbook that links. Often it is faster to open a workbook than to read the links from a closed workbook.

Virus scanner settings.

Some virus scanner settings can cause problems or slowness with open/close/save, particularly on a server. If you think this might be the problem, try temporarily switching the scanner off.

Slow Calculation causing slow open/save

Under some circumstances Excel will recalculate your workbook when it opens or saves it. If the calculation time for your workbook is long and this is causing a problem make sure you have calculation set to manual, and that consider turning off calculate before save (Tools-->Options-->Calculation).

Addins/VBA Auto-Open problems.

See if the workbook opens faster when you open it but **hold the shift key down**. This disables any macros in the workbook.

Too many files in XLStart or the Alternate Startup Directory.

If you have a large number of files in your XLStart directory, or an alternate startup directory (Tools-->Options-->General), this may slow Excel down.

Too many Add-Ins.

I tend to use a lot of add-ins, and if I don't check from time to time the number of add-ins automatically loaded can get large. Periodically prune the list of add-ins that are automatically loaded (Tools-->Addins), or check out my Auto-Reversioning Demand-Loading Addin Loader.

Early version of Excel97

The original version of Excel 97 had a number of problems that could cause slow opening/lockup. Make sure your installed version is at the SR2 level (Help -->About should say Microsoft ® Excel 97 - SR2). If you have an earlier version contact Microsoft for a free upgrade CD, or download the updated from the Microsoft site.

Excel 2000.

- The original version of Excel 2000 was very slow at updating links to closed Excel workbooks. Make sure you are at the SR1 level.
- When in Automatic calculation mode Excel 2000 calculates workbooks last saved by Excel97 as they are opened. Where possible make sure all your workbooks have been saved by Excel 2000, or use manual calculation. See [MSKB Q210162](#)

Excel 2002

When Excel 2002 opens a workbook that was last saved by a previous version of Excel substantial extra work is done to calculate and error check the workbook. Where possible make sure all your workbooks have been saved by Excel 2002, and avoid having some users on previous Excel versions.

If you have upgraded to Excel2002 from an earlier version and you have Norton Antivirus installed the integrated virus checking may cause slower file opening.

Excel 2002 has advanced encryption methods that may slow open and close for protected workbooks.

New Excel 2002 features such as Fast Search and Background Error Checking may also slow response time.

A large number of VLOOKUPs may cause slow opening with Excel2002: see MSKB 327365

WinXP Pro SP1 with Windows2000 Server: Office XP slow file retrieval and crashes

There are frequent reports of slow opening with Office XP/WinXP. One possible solution is suggested to be SMB signing incompatibility: see

<http://asia.cnet.com/itmanager/netadmin/0,39006400,39108281,00.htm>

<http://support.microsoft.com/default.aspx?scid=kb;EN-US;810907>

Toolbar Files (.XLB).

Check the size of your toolbar file. Mine is about 12KB.

Adding/changing or customising toolbars causes the size of your toolbar.XLB file to increase. Each user has their own XLB file. Deleting it removes all your toolbar customisations (renaming it to toolbar.OLD is safer): A new .XLB file will be created the next time you open Excel.

You can find your XLB files by using Windows search for *.xlb.

LAN or WAN problems.

The complex structure of an Excel workbook means that Excel does not read or write a workbook sequentially. In addition Excel SAVE's a workbook first as a temporary file, then deletes the workbook, and then renames the temporary file. This process is timing dependent and may contribute to opening/closing bottlenecks or problems, particularly with a Server or Virus scanner. Sometimes you are better off copying a workbook from the Server to your local PC, operating on it locally, and then copying it back.

See MSKB Q254733 for WAN problems with Forms.

Excel Calculation Bottlenecks

These are some of the most frequently occurring Excel calculation bottlenecks.

Repeated or duplicated calculations

Because most spreadsheets are constructed by copying formulae containing a mixture of absolute and relative references it is very easy to create a large number of formulae containing repeated or duplicated calculations and references.

The golden rule for removing calculation bottlenecks is to look for these repeated/duplicated calculations, move them to a single 'helper' cell, and then reference the helper cell from the original formulae.

Links

My advice is to avoid Workbook links wherever possible.

Workbook Links:

Avoid Workbook links wherever possible:

Links are slow, easily broken, error-prone and not always easy to find and fix. If Excel tells you that your workbook has links (phantom links), but you can't find them, download Bill Manville's excellent FINDLINK.XLA from Stephen Bullen's website <http://www.bmsltd.ie/>.

Fewer larger Workbooks are usually (but not always) better than many smaller Workbooks: the exception is usually when you have a lot of front-end calculations that are very rarely recalculated so it makes sense to put them in a separate workbook, or when you don't have enough real memory.

Excel calculates Workbooks in Workbook name sequence, so make sure the inter-workbook references only go in one direction (no forward links).

Try to use simple direct cell references that work on closed Workbooks. That way you can avoid recalculating ALL your linked Workbooks whenever you recalculate ANY Workbook, and you can see the values Excel has read from the closed Workbook, which is often important for debugging and auditing the Workbook.

Worksheet Links:

Using fewer worksheets usually calculates faster, but you need to balance this against maintainability and useability.

Using lots of Worksheets can make your Workbook easier to use, but there can be a large performance and memory penalty, particularly if you have complex inter-linking between Worksheets.

Excel 97 and 2000 calculate Worksheets in Worksheet name sequence, so try and make sure that references between sheets always refer backwards to sheets that have already been calculated. Forward Worksheet cross-references to sheets that have not yet been calculated can be very slow.

Excel 2002 and 2003 handle inter Worksheet dependencies in a more global way, and can be substantially faster on problem Workbooks.

See **FastExcel Optimise Worksheet Sequence** to automatically determine a near-optimum Worksheet calculation sequence.

- A good design compromise is to try to split large workbooks as follows:
- A worksheet for What-If parameters (Dashboard).
- A worksheet with your more-or-less static data on it.
- As few as possible worksheets for calculations.
- A worksheet for reports.
- A worksheet for summary results.

Small workbooks may combine the reports and summary reports sheets, and the parameters and static data sheets.

Exact Match Lookup using VLOOKUP, HLOOKUP, MATCH

Exact Match Lookup is slow, approximate match Lookup is fast.

See Optimising Lookups Tips and Tricks and FastExcel's AVLOOKUP.

If you are doing Lookup's using the exact match option for these functions the calculation time for the function is proportional to the number of cells scanned before a match is found.

For lookups over large ranges this time can be very significant.

Lookup time using the approximate match options of **VLOOKUP**, **HLOOKUP**, **MATCH** on sorted data is fast and is not significantly increased by the length of the range you are looking up. (Characteristics are the same as Binary Search).

Array Formulae:

Array formulae are very powerful, but need to be handled with care. Make sure you are using the smallest possible ranges.

Single-cell array formulae are evaluated multiple times, depending on the number of cells referred to in the formula. This can take significant time, and may or may not be faster than the alternatives.

It is important to minimise the number of cells referenced by array formulae.

You may be able to use Excel's DSUM function or FastExcel's AVLOOKUPS function to speed up conditional sum array functions.

A range array formula covering multiple cells may be faster to calculate than individual formulae in each cell (although the speed advantage seems less in Excel97 and Excel2000). Multi-cell array formulae also make smaller files. Because array formulae and functions like SUM which reference ranges influence the sequence in which Excel calculates, you should try and avoid mixing row and column references or overlapping array references, for example:

Adding SUM functions at the foot of each column to large sets of single-row, multiple column array functions can sometimes cause calculation time to increase by a factor of 2 or more.

Functions

SUM, SUBTOTAL, COUNT, SUMIF and COUNTIF.

Don't SUM unnecessary cell that contain data/results..

The calculation time for these functions is proportional to the number of used cells you are summing or counting, so make sure you don't include more used cells than you need. See Dynamic Ranges.

User-Defined Functions:

UDF's can be slow.

It's usually significantly faster to use Excel's Worksheet functions and calculations than to use User-Defined functions. This is because there is significant overhead transferring information from Excel to the UDF and back.

Volatile Functions:

Avoid using Volatile functions if possible.

Usually Excel's dependency recalculation engine will only calculate only the minimum number of cells. If you have many Volatile functions the recalculation time can increase to close to the full calculation time, or even exceed it in extreme cases

The **FastExcel** Profile Workbook command shows you the degree of Volatility of your whole Workbook and of each worksheet.

Other:

Conditional Formats & Data Validation:

Conditional Formats and Data Validation are great, but using a lot of them will significantly slow down calculation.

FastExcel shows you the number of the conditional formats used.

Defined Names:

Defined Names are one of Excel's most powerful features, but they do take some additional calculation time. Using Names that refer to other Worksheets adds an additional level of complexity to the calculation process. Also try to avoid nested names (names referring to other names).

Because Names are calculated **every time** a formula that refers to them is calculated you should avoid putting calculation-intensive formulae or functions in defined names. It can be significantly faster in these cases to put your calculation-intensive formula or function in a spare cell somewhere and refer to that cell instead, either directly or via a name.

FastExcel shows you the number of defined names used.

Character Manipulation in Formulae:

Manipulating character strings in formulae is relatively slow. When you have a lot of formulae handling character strings you may notice this:

Rob Bovey recommends using IF(ISBLANK(A1),...) to test if a cell is blank rather than IF(A1="",...).

Concatenating two columns/rows to make a new column/row is slow.

Size and Memory Problems.

Here are some of the factors that can cause workbook size and memory problems.

Used Range Problems.

Used Range problems are a frequent cause of file bloat.

Excel attempts to track the last used cell on each worksheet, but does not always reset this as you might expect. The last used cell is not just the last cell that has contained a formula or data, but also the last cell with formatting (includes non-standard column width). Sometimes the last cell somehow gets set thousand of rows or hundreds of columns too large, either because of excess formatting or because a large number of cells, rows or columns have been inserted or deleted. This can cause the file size and memory used to grow very large.

You can check the location of the last visible used cell on a worksheet using Ctrl-End or Edit-->GoTo-->Special-->LastCell.

If you have hidden cells or rows it is a good idea to unhide them before doing the Ctrl-End.

The **FastExcel Profile Workbook** command will show you, for each worksheet, how many cells Excel thinks are being used, and the percentage of these (%Waste) that are outside the last formula or data cell.

You can correct the Used Range problem by deleting the unwanted rows and columns and saving the workbook.

Make sure you backup your workbook before the deletion in case you have formulae referencing the deleted cells:

If you have formulae which reference the rows and columns you have deleted because Excel will try to adjust the references in these formulae. If the cells referred to no longer exist the reference becomes #N/A.

In Excel97 and later accessing the UsedRange property of a worksheet causes Excel to reset the used range.

FastExcel Version 2 contains a Clean Workbook command that allows you to clean the used range in a non-destructive way.

Tracking Changes in a Shared Workbook.

This will rapidly cause your Workbook file-size to increase.

Saving in Dual format: Excel 97 & 5.0/95.

This will double the size of your file.

All **FastExcel** Profile ... commands will highlight this format in orange.

InterSheet References.

Intersheet references use more memory than onsheet references. References to ranges on other worksheets should be as small as possible because Excel can use considerable memory storing intersheet dependency trees.

VBA Changes.

I recommend Rob Bovey's Code Cleaner.

Making changes to any Visual Basic Modules contained in your Workbook will tend to accumulate redundant material that will grow the size of the Workbook. Use Rob Bovey's Code Cleaner regularly to eliminate this:

<http://www.appspro.com/>

Charts, Graphics, Zoom etc.

Excel will only use a limited amount of memory on Charts, Graphic Objects and Zoom before you get an “Out of Memory” message, regardless of how much RAM you have.

Importing bitmaps.

To minimise memory and workbook size, make sure you have resized your bitmaps to the size you want BEFORE inserting them into Excel.

Zero-Sized Graphics.

Some conditions cause Excel to make Graphic Objects visibility zero height and/or width. When this happens the graphic objects still use memory and filesize, but these are hard to find and delete except using VBA.

External Links.

Excel will only use a limited amount of memory on External Links before you get an “Out of Memory” message, regardless of how much RAM you have.

Formats.

Using many different formats will increase the size of your Workbook.

Opening a Lotus WK4 file.

When you open a Lotus WK4 file and save it as an Excel file, the file may get very large. This is caused by the import process assigning individual formats to each cell in a column.

See MSKB article Q123269 for more details.

Multi-Cell Array Formulae

When you have a large rectangular block of formulae you may be able to convert them to a single multi-cell array formula. This will reduce file size somewhat, but does not appear to significantly affect memory.

Toolbar Files (.XLB).

Adding/changing or customising toolbars causes the size of your toolbar.XLB file to increase. Each user has their own XLB file. Deleting it removes all your toolbar customisations: a new .XLB file will be created the next time you open Excel.

Check the size of your toolbar file. Mine is about 12KB.

Optimising Tips and Tricks

Here are a number of Tips and Tricks you can use to optimise calculation speed.

Prioritise the Calculation Bottlenecks

Use FastExcel to determine where the calculation bottlenecks are and how much calculation time each one is using. Then start with the largest!

Count the number of cells references and calculations require

Its not the number of formulae that consumes the calculation time, it's the number of cell references and calculation operations.

Work out roughly how many cell references and calculations are required for Excel to calculate the result for this bottleneck.

Then think how you might get the same result with fewer references and calculations.

Look for repeated/duplicated expressions

Move repeated calculations out to a 'helper' cell.

Avoid complex mega-formulae: keep it simple and use more formulae: its generally faster to calculate.

Look at the bottleneck and see if you can spot expressions or calculations or references to ranges of cells that are repeated or duplicated, either within the bottleneck formulae or across a number of bottleneck formulae.

Then work out how to eliminate the duplication by moving that part of the formulae to a separate 'helper' cell or cells, and replace it in the bottleneck formulae with a reference to the 'helper' cell.

Volatile Functions

Volatile functions can slow down recalculation because they increase the number of formulae that must be recalculated at each calculation.

You can often reduce the number of volatile functions by using INDEX rather than OFFSET, and CHOOSE rather than INDIRECT.

But OFFSET itself is a fast function and can often be used in creative ways that give very fast calculation.

Lookup Options

Lookups are often the most significant factor in calculation times, and fortunately there are many ways of improving Lookup calculation time.

With FastExcel Version 2 you can now use the AVLOOKUP function, which is faster than VLOOKUP and INDEX/MATCH in many circumstances.

Make sure you have understood the options in MATCH, VLOOKUP and HLOOKUP.

MATCH(lookup value, lookup array, match type)

- Matchtype=1 returns the largest match less than or equal to lookup value if lookup array is sorted ascending. This is the default
- Matchtype=0 requests an exact match
- Matchtype=-1 returns the smallest match greater than or equal to lookup value if lookup array is sorted descending

VLOOKUP(lookup value, table array, colnum, range-lookup)

- Range-lookup=TRUE returns the largest match less than or equal to lookup value. This is the default option. Table Array MUST be sorted ascending.
- Range-lookup=FALSE requests an exact match. Table array does not need to be sorted.

Avoid using exact match lookup if possible.

If you are doing Lookup's using the exact match option the calculation time for the function is proportional to the number of cells scanned before a match is found. For Lookups over large ranges this time can be very significant.

Lookup time using the approximate match options of **VLOOKUP**, **HLOOKUP**, **MATCH** on sorted data is fast and not significantly increased by the length of the range you are looking up. (Characteristics are the same as Binary Search).

VLOOKUP versus INDEX and MATCH or OFFSET.

I recommend using INDEX and MATCH.

VLOOKUP is slightly faster (approx. 5%), simpler and uses less memory than a combination of **MATCH** and **INDEX** or **OFFSET**.

However the additional flexibility offered by **MATCH** and **INDEX** often allows you to make significant timesaving compared to **VLOOKUP**.

INDEX is very fast and from Excel 97 onwards is a non-volatile function (speeds up recalculation).

OFFSET is also very fast, but it's a Volatile function, and sometimes causes a significant increase in the time taken to process the dependency trees.

Converting VLOOKUP to INDEX and MATCH.

These statements return the same answer:

```
VLOOKUP (A1, Data!$A$2:$F$1000, 3, False)
```

```
INDEX (Data!$A$2:$F$1000, MATCH (A1, $A$1:$A$1000, 0) , 3)
```

Speeding up Lookup's

Because exact match lookups are so slow it's worth looking for ways of speeding things up:

Use FastExcel's AVLOOKUP function:

AVLOOKUP is significantly faster than VLOOKUP in many circumstances.

Use One Worksheet.

If speed is critical keep lookups and data on the same sheet.

Keep exact match lookups on the same Worksheet as the data they are looking up: It's significantly faster.

Use Excel 2000:

Upgrade to Excel 2000.

Excel 2000 is significantly faster than Excel 97 for exact matches. But make sure you have installed the SR1 Service Release that fixes a problem with Lookups into closed workbooks (see MSKB Q248173).

SORT the Data Whenever Possible.

SORT your data and use approximate Match.

Whenever possible SORT the data first, (SORT is very fast) and use approximate match.

Minimise the Range of Cells you are Looking Up.

The smaller the range the better.

When doing exact match lookups restrict the range of cells to be scanned to a minimum. Use dynamic range names rather than referring to a very large number of rows or columns. Sometimes you can pre-calculate a lower and upper range limit for the lookup.

Sorted Data with Missing Values.

Two approximate lookups are usually faster than one exact lookup.

If you can sort your data but still cannot use approximate match because you can't be sure that the value you are looking up exists in the lookup range, then try this:

```
IF(lookup_val=Index(lookup_array,MATCH(lookup_val,lookup_list),1),Index(lookup_array,MATCH(lookup_val,lookup_array), colnum),"notexist")
```

This does an approximate lookup on the lookup list, and if the lookup value = the answer in the lookup column you have found an exact match, so redo the approximate lookup on the column you want, otherwise it's a missing value. Note that you need to make sure you never lookup a value smaller than the smallest value in the list, possibly by adding a dummy list entry.

Two approximate matches are significantly faster than one exact match for a lookup over a large number of rows (breakeven point is about 10-20 rows).

FastExcel's AVLOOKUP function allows you to handle the missing value problem very efficiently.

Store the result of an exact MATCH and reuse it.

Unsorted Data with Missing Values.

If you have to use exact match lookup on unsorted data and you can't be sure that the lookup value exists you often have to handle the #N/A that gets returned if no match is found.

The simplest and slowest way is to use an IF function containing two lookups:

```
IF (ISNA (VLOOKUP (lookupval, table, 2, FALSE)) , 0,
VLOOKUP (lookupval, table, 2, FALSE))
```

You can avoid the double exact lookup if you use exact MATCH once, store the result in a cell, and then test the result before doing an INDEX:

In A1 =MATCH (lookupvalue, lookuparray, 0)

In B1 =IF(ISNA(A1),0,INDEX(tablearray,A1,colnum))

If you cannot use two cells then use COUNTIF, on average it is faster than an exact match lookup:

```
IF (COUNTIF (lookuparray, lookupvalue)=0, 0,
VLOOKUP (lookupval, table, 2, FALSE))
```

FastExcel's AVLOOKUP function allows you to handle the missing value problem very efficiently

You can often reuse a stored exact MATCH many times.

Exact Match Lookups on Multiple Columns.

If you are doing exact lookups on multiple result columns you can save a lot of time using one MATCH and many INDEX statements rather than many VLOOKUPs.

Add an extra column for the MATCH to store the result (stored_row)

For each column use

```
INDEX (Lookup_Range, stored_row, column_number)
```

Alternatively you can use VLOOKUP in an array formula:

```
{VLOOKUP (lookupvalue, {4, 2}, FALSE) }
```

You can also return many cells from one lookup operation.

Looking Up a Set of Contiguous Rows or Columns.

If you want to lookup a number of contiguous columns then you can use INDEX in an array formula to return multiple columns at once (use 0 as the column number). You can also use INDEX to return multiple rows at once.

```
{ INDEX ($A$1 : $J$1000, stored_row, 0) }
```

This returns columns A to J in the stored row created by a previous MATCH

Looking Up a Rectangular Block of Cells.

You can use MATCH and OFFSET to return a rectangular block of cells.

Two-Dimensional Lookup

Multi-dimensional lookup can also be done efficiently.

Two-dimensional table lookup using separate lookup's on the rows and columns of a table can be efficiently done using an INDEX with two embedded MATCH functions.

FastExcel's AVLOOKUP function allows you to handle two-dimensional lookups efficiently.

Multiple-Index Lookup

In large spreadsheets you often need to lookup using multiple indexes, such as looking up product volumes in a country.

The simple way to do this is to concatenate the indexes and lookup using concatenated lookup values. This is inefficient for two reasons:

- Concatenating strings is a calculation-intensive operation.
- The lookup will cover a large range.

It is often more efficient to calculate a subset range for the lookup: for example by finding the first and last row for the country, and then looking up the product within that range.

The FastExcel Version 2 AVLOOKUP function has built-in methods that easily and efficiently handle multiple-index lookup.

Three-dimensional lookup.

If you need to lookup the table to use as well as the row and the column here are some techniques you can use, focusing on how to make Excel lookup/choose the table.

If each table you want to lookup (the third dimension) is stored as a set of range names, or as a table of text strings that represent ranges, then you may be able to use INDIRECT or CHOOSE.

Using CHOOSE and range names can be a very efficient method, and it is not volatile, but it is best suited to only a small number of tables:

```
INDEX(CHOOSE(TableLookup_Value, TableName1, TableName2, TableName3, TableName4), MATCH(RowLookup_Value, $A$2:$A$1000), MATCH(colLookup_value, $B$1:$Z$1))
```

The example above dynamically uses TableLookup_Value to choose which range name (TableName1, TableName2, ...) to use for the lookup table.

```
INDEX(INDIRECT("Sheet" & TableLookup_Value & "!$B$2:$Z$1000"), MATCH(RowLookup_Value, $A$2:$A$1000), MATCH(colLookup_value, $B$1:$Z$1))
```

This example uses INDIRECT and TableLookup_Value to dynamically create the sheet name to use for the lookup table. This method has the advantage of being simple and can handle a large number of tables, but because INDIRECT is a volatile function the lookup will be calculated at every calculation even if none of the data has changed.

You could also use VLOOKUP to find the name of the sheet or the text string to use for the table, and then use INDIRECT to convert the resulting text into a range:

```
INDEX(INDIRECT(VLOOKUP(TableLookup_Value, TableOfTables, 1)), MATCH(RowLookup_Value, $A$2:$A$1000), MATCH(colLookup_value, $B$1:$Z$1))
```

Another technique is to aggregate all your tables into one giant table, but with an additional column that identifies the individual tables. You can then use the techniques for multiple-index lookup above.

Wildcard Lookup

AVLOOKUP, AMATCH, MATCH, VLOOKUP and HLOOKUP allows you to use the wildcard characters ? (Any single character) and * (no character or any number of characters) on alphabetic exact matches. Sometimes this can avoid multiple matches.

Use Dynamic Ranges to minimise the number of cells that Excel has to calculate.

Dynamic Ranges

See FastExcel V2 Dynamic Range Wizard for easy creation of dynamic ranges.

When you often have to add extra rows or columns of data to your spreadsheets, you need to find a way of having your Excel formulae automatically refer to the new data area.

You can do this by using a large range in your formulae that extends well beyond your current data boundaries, for instance by referring to a whole column. However this causes very inefficient calculation.

The solution is to use dynamic named ranges. By using the OFFSET and COUNTA functions in the definition of a named range, the area that the named range refers to can be made to dynamically expand and contract. For example create a defined name as:

=OFFSET(Sheet1!\$A\$1,0,0,COUNTA(Sheet1!\$A:\$A),1)

When you use the dynamic range name in a formula it automatically expands to include new entries.

The benefit of this (apart from the fact that you don't have to change all your formulae) is that Excel then only has to calculate using exactly the right number of rows and columns. However there is a performance hit if you use a lot of Dynamic Ranges.

The main performance hit is because OFFSET is a volatile function and therefore is always recalculated, and because the COUNTA function inside the OFFSET has to look at a large number of rows. You can minimise this performance hit by storing the COUNTA part of the formula in a separate cell, and then referring to the cell in the dynamic range:

Counts!z1=COUNTA(Sheet1!\$A:\$A)

DynamicRange=OFFSET(Sheet1!\$A\$1,0,0,Counts!\$Z\$1,1)

You can also use functions like INDIRECT to construct dynamic ranges.

The FastExcel sample problem makes extensive use of dynamic range names, and also shows how to optimise them. It contains two versions of the optimised workbook, one with dynamic names, and one without.

FastExcel Version 2 has a Dynamic Range Wizard that makes it easy to create dynamic range names, and a several new counting functions that overcome the limitations of the COUNTA function.

For further explanation of Dynamic Ranges see:

- Chip Pearson's page on named and dynamic ranges at Pearson Software Consulting with a downloadable sample workbook.
- Rodney Powell's Beyond Technology pages at <http://www.beyondtechnology.com/geeks007.shtml>.

Generating Workbooks.

Dynamically generating the formulae in workbooks from database tables can produce workbooks that calculate extremely fast, because it is often possible to avoid using many time-expensive Excel features such as Lookups.

Build Time versus Run-Time:

Many Workbooks contain a significant number of formulae and lookups that are concerned with getting the input data into the right shape for the calculations, or are being used as defensive measures against changes in the size or shape of the data.

These things do not change often, so can either be put in a separate spreadsheet that does not need to be recalculated, or you can use copy paste special values to eliminate the formulae.

FastExcel allows you to use extended calculation modes so that you can have some worksheets in manual calculation mode, and some worksheets in automatic calculation mode. With FastExcel you can also make Excel only calculate the active workbook when multiple workbooks are open.

Totals and Conditional Sums

Totals as Final Results

If you need to produce totals and subtotals as part of the final results of your Workbook try using Pivot tables.

Totals as Intermediate Results

Pivot tables are a great way to produce summary reports, but try to avoid creating formulae that reference Pivot Table results.

If you need to use totals and sub-totals as intermediate results in your calculation chain then it is not advisable to use Pivot tables unless you can ensure that:

- The Pivot table has been refreshed correctly during the calculation.
- The Pivot table has not been changed so that the information is still visible

If you still want to use Pivot tables as intermediate results then use the GETPIVOTDATA function.

Period-to-date and cumulative SUMs

There are two methods of doing period-to-date or cumulative SUMs:

Suppose the numbers you want to cumulatively SUM are in column A and you want Column B to have the cumulative sum.

You can create a formula in column B like =SUM(\$A\$1:\$A2) and drag it down as far as you need. The beginning cell of the SUM is anchored in A1, but because the finishing cell has a relative row reference it will automatically increase for each row.

Or you can create a formula like =\$A1 in cell B1 and like =\$B1+\$A2 in B2 and drag it down as far as you need. This calculates the cumulative cell by adding this rows number to the previous cumulative SUM.

For 1000 rows the first method makes Excel do about 500000 calculations, but the second method only makes Excel do about 2000 calculations (250 times more efficient!).

SUM and SUMIF:

Excel's SUM and SUMIF functions are frequently used over a large number of cells. Calculation time for these functions is proportionate to the number of cells covered.

Subset Addressing

When you have multiple sorted indexes to a table (for instance Site within Area) you can often save significant calculation time by dynamically calculating the address of a subset range of rows (or columns) to use in the SUM or SUMIF. See SUMIF Example or the FastExcel Sample problem for examples of this technique.

Wildcard SUMIF & COUNTIF

You can use the wildcard characters ? (Any single character) and * (no character or any number of characters) as part of the SUMIF & COUNTIF criteria on alphabetic ranges.

DSUM

Excel's DSUM function is very fast and uses a criteria range external to the function to specify multiple conditions for the sum. These criteria ranges allow

great flexibility, but are difficult to use and maintain for large numbers of DSUM functions.

Subtotals

Embedded Subtotals:

You can create embedded subtotals using the Data → Subtotals command on sorted data. I do not recommend using embedded subtotals because of the danger of inadvertent double counting.

The SUBTOTAL function.

Use the SUBTOTAL function to SUM filtered Lists.

The SUBTOTAL function is useful because, unlike SUM, it ignores:

- Hidden rows that result from filtering a list
- In Excel 2003 only you can also make SUBTOTAL ignore all hidden rows, not just filtered out rows.
- Other SUBTOTAL functions.

Combined Subtotals and Totals.

If you are working with multidimensional data tables you often need to reference both subtotal and total intermediate results. Make sure you calculate the totals as a SUM of the subtotals rather than as a SUM of the whole table.

It's not the SIZE of a Workbook that consumes calculation time, it's the interpretation, calculation and formatting.

Sometimes it's better to have more rows and columns and less complex calculations. It's also often easier to understand.

Multi-Level and Repeated Calculations:

Many Workbooks are built using large formulae containing repeated calculations.

Try using additional rows and columns to store calculate intermediate results ONCE and re-use them in other formulae.

For example you may have a time-expensive formula where you want the result of an error to be shown as zero. You can write this as:

Either

- a single formula, which is slow:

```
B1=IF(ISERROR(time expensive formula),0,time  
expensive formula)
```

Or

- two formulae, which is fast:

```
A1=time expensive formula
```

```
B1=IF(ISERROR(A1),0,A1)
```

Large Workbooks are frequently dealing with information at different levels (areas within countries within continents, or products within area within region...)

In these cases try to move calculations from the most detailed level to a higher level.

Array Formulae

Array formulae are one of Excel's most powerful features, although not always the easiest to use. Look at Bob Umlas's excellent paper at <http://www.emailoffice.com/excel/arrays-bobumlas.html> for many examples of how to use array formulae.

Single-cell array formulae are evaluated multiple times, depending on the number of cells referred to in the formula. This can take significant time, and may or may not be faster than the alternatives. A range array formula covering multiple cells may be faster to calculate than individual formulae in each cell (although the speed advantage seems less in Excel97 and Excel2000).

Because Array formulae and functions like SUM which reference ranges influence the sequence in which Excel calculates, you should try and avoid mixing row and column references or overlapping array references.

Minimise the Number of Array Operations

Minimise the number of cells and expressions evaluated.

Use helper columns or rows to take as many cell references as possible out of the array formulae.

The main key to optimising the calculation speed of array formulae is to make sure that the number of cells and expressions evaluated in the array formula is as small as possible.

Remember that an array formula is a bit like a volatile formula: if any single one of the cells it references has changed or is volatile or has been recalculated then the array formula **will evaluate ALL the cells it references**.

- Take expressions and range references out of the array formulae into separate helper columns/rows. This will make much better use of Excel's smart recalculation process.
- Don't reference complete rows, or more rows and columns than you need.
- Use dynamic range names where possible. Even though they are volatile it's worthwhile because they minimise the size of the ranges.
- Be careful with array formulae that reference both a row and a column: this forces the calculation of a rectangular range.
- Use SUMPRODUCT if possible: it's slightly faster than the equivalent array formula.

Array formulae SUM with multiple conditions

One frequent use of Array formulae is to do a sum with multiple conditions. This is relatively easy to do, particularly if you use Excel's Conditional Sum Wizard, but often very slow. Usually there are alternative ways of getting the same result, but much faster.

Two good methods of speeding this up are:

Split out the multiple conditions into a column of helper formulae that return True or False for each row, then reference the helper column in a SUMIF or array formulae. Whilst it may seem that for a single array formula that this does not reduce the number of calculations, in fact most of the time this allows Excel's smart recalculation process to only recalculate the formulae in the helper column that NEED to be recalculated.

If the data can be sorted then a good technique is to count groups of rows and limit the array formulae to looking at the subset groups.

See [SUMIF example](#) for a worked example of this. The downloadable [FastExcel Sample Problem](#) also has an example, which shows the method and the considerable time saving which is often achievable.

Using SUMPRODUCT for multiple condition array formulae.

There are some advantages to using SUMPRODUCT rather than SUM array formulae:

- SUMPRODUCT does not have to be array-entered using Control-shift-enter.
- SUMPRODUCT is usually slightly faster (5-10%).
- SUMPRODUCT can be made to treat blanks and text as zero.

You can use SUMPRODUCT for multiple condition array formulae as follows:

`SUMPRODUCT(--(Condition1),--(Condition2),RangetoSum)`

where Condition1 & 2 are conditional expressions such as `A1:A10000<=$Z4`. Because conditional expressions return True or False rather than numbers they need to be coerced to numbers inside the SUMPRODUCT function. You can do this using two minus signs (--) or by adding 0 (+0) or by multiplying by 1 (*1). Using -- is very slightly faster than +0 or *1.

Note that the size and shape of the ranges or arrays used in the conditional expressions and range to sum must be the same, and cannot contain entire columns.

You can also directly multiply the terms inside SUMPRODUCT rather than separate them by commas:

`SUMPRODUCT((Condition1)*(Condition2)*RangetoSum)`

But this is usually slightly slower than using the comma syntax, and gives an error if the range to sum contains a text value.

However it is slightly more flexible in that the range to sum may have for instance multiple columns when the conditions only have 1 column.

Using SUMPRODUCT to multiply and add ranges and arrays.

In cases like weighted average calculations where you need to multiply a range of numbers by another range of numbers and sum the results using the comma syntax for SUMPRODUCT can be 20-25% faster than an array entered SUM:

`{=SUM(D2:D10301*E2:E10301)}`

`=SUMPRODUCT(D2:D10301*E2:E10301)`

`=SUMPRODUCT(D2:D10301,E2:E10301)`

These three formulae all produce the same result, but the third formula using the comma syntax for SUMPRODUCT only takes about 77% of the time to calculate that the other two formulae need.

Use DSUM instead of multiple condition array formulae.

If you only have a few array formulae sums with multiple conditions you may be able to use DSUM instead. DSUM is significantly faster than equivalent array formulae. The disadvantage of DSUM is that the criteria have to be in a separate range, which makes it impractical to use and maintain in many circumstances.

Use FastExcel AVLOOKUPS.

You can use the FastExcel Version 2 AVLOOKUPS function inside a SUM function to return multiple rows and SUM them. If you are doing multiple conditional sums, particularly on sorted data, AVLOOKUPS can be significantly faster than array formulae

Array and Function Calculation Bottlenecks

Excel's calculation engine is optimised to exploit array formulae and functions that reference ranges. However some unusual arrangements of these formulae and functions can sometimes, but not always, cause significantly increased calculation time.

If you find a calculation bottleneck involving array-formulae and range functions look for:

- Partially overlapping references:
- Array formulae/range functions which reference part of a block of cells that are calculated in another array formula/range function. This situation can frequently occur in time series analysis.
- One set of formulae referencing by row, and a second set of formulae referencing the first set by column:
- A large set of single-row array formulae covering a block of columns, with SUM functions at the foot of each column.

Advantages and Disadvantages of Array Formulae

Plus points of Array Formulae:

- Array Formulae are concise: You can eliminate columns and rows by packing the calculations into an array formula.
- Array Formulae are powerful: You can easily perform many complex calculations such as multiple conditional sums and counts using array formulae.
- Array Formulae save disk space: Using array formulae covering a range of cells can reduce workbook size compared to equivalent individual formulae. Memory used does not usually seem to reduce significantly.
- Array Formulae offer increased protection: Excel will only allow you to alter a complete array formula block so the user is prevented from accidentally changing a single formula.

Minus points of Array Formulae:

- The Black-Box effect: Array Formulae can be complex and hard to understand. Most Excel users do not understand array formulae at all. This can reduce confidence and usability of your spreadsheet.
- Calculation Overhead: Each time an array formula is calculated **all** of the virtual cells needed by the array formula are calculated, regardless of whether this is required or not. This may cause the array formula to be slower than a non-array equivalent set of formulae.
- Requirement for all components of the array formulae to be the same size: This may require the array formula to perform a large number of unnecessary calculations. This problem frequently occurs in a SUM with multiple conditions problem. Calculating the subset range that needs to be summed may often make significant calculation speed improvements. The example below shows how several hours of Array Calculations can be reduced to under a second.

Example: SUM with Multiple Conditions

The SUMIF function only allows a single condition. Often you need to do SUM calculations that are dependent on multiple conditions. These calculations can be very sensitive to the method of calculation used.

This example shows how to reduce the calculation time for this problem from several hours to a tenth of a second.

The example below shows an iterative improvement approach.

The different approaches have calculation times ranging between 3 hours to 0.14 seconds and each approach gives identical results.

- A number of Sites act as origins and destinations in a network.
- There are 10000 origin-destination pairs called **Routes**.
- Each Route has a shipment volume for each of 5 years.
- Each origin and destination is graded as A or B.
- Each origin Site always has additional destinations that start with OTH.

We need to calculate a coverage factor for each route for each of the 5 years as follows:

Factor for a route=(sum of volumes for all the routes starting at this routes origin and with both origin and destination=A)/(total volume for this routes origin – volumes from this routes origin to destinations starting with OTH)

The Conditional Sum Wizard is easy to use but produces slow calculations.

Over 3 hours.

The easiest approach is to use Excel's built-in Conditional Sum Wizard (you can install this using Tools→Addins).

The Conditional Sum Wizard generates an array formula using nested Ifs, for example:

```
{SUM(IF($A$4:$A$10003=$A4,IF(L$4:L$10003="AA"),D$4:D$10003,0),0)}
```

This calculation for the 10000 routes takes 744 seconds on my 500 MHZ laptop.

To solve the problem would need another 15 similar columns of formulae, and the calculation time would be over 3 hours.

Using an Array formula is faster but still too slow.

An alternative approach is to use multiplicative AND's in the array formula:

```
{SUM(($A$4:$A$10003=$A4)*(L$4:L$10003="AA")*(D$4:D$10003))}
```

This calculation is slightly faster at 723 seconds for a single column against 744, but still far too slow.

Or you can use SUMPRODUCT, which does not need to be array-entered:

```
SUMPRODUCT(--($A$4:$A$10003=$A4),--(L$4:L$10003="AA"),D$4:D$10003)
```

SUMPRODUCT is faster than the array formula, 711 seconds compared to the 723, but still too slow.

Calculate once, store the results and then lookup the results many times.

89 Seconds.

With this calculation all routes with the same origin have the same answer: so we can reduce the calculation time significantly by only doing it once for each of the 105 unique origins, and then looking up the answer for each of the 10300 routes.

The following formulae do the calculation for one year at Origin level in 17.8 seconds, or 5 years in 89 seconds. This is much better but still too slow.

```
{SUM((LaneCalc!$A$4:$A$10003=$A5)*
(LaneCalc!$L$4:$L$10003="AA")*(LaneCalc!D$4:D$10003))}

SUM((LaneCalc!$A$4:$A$10003=$A5)*
(LEFT(LaneCalc!$B$4:$B$10003,3)="OTH")*
(LaneCalc!D$4:D$10003))

SUMIF(LaneCalc!$A$4:$A$10003,A5,LaneCalc!D$4:D$10003)

E5/(G5-F5)
```

Dynamically calculate subset blocks to be used in the calculation.

0.7 seconds

These formulae do not exploit the fact that the routes are sorted by destination within origin, so that the block of rows we need to do the SUM over for each origin is contiguous and much smaller than 10000 rows. However we need to be aware that the origin and destination Sites could change.

We can calculate the range to use in the formulae using three extra columns:

The number of routes for this origin (numrows) is:

```
COUNTIF(LaneCalc!$A$4:$A$10003,A5)
```

The row number of the last route for this origin is the start row for this origin plus the number of routes.

The row number of the first route for the next origin (startrow) is the previous origin's last route row number plus 1.

The coverage formulae then do not need array formulae and can use SUMIF and OFFSET as follows:

```
SUMIF(OFFSET(LaneCalc!$K$1,startrow,1,numrows,1),
"AA",OFFSET(LaneCalc!$C$1,startrow,1,numrows,1))

SUMIF(OFFSET(LaneCalc!$B$1,startrow,0,numrows,1),
"oth*",OFFSET(LaneCalc!$C$1,startrow,1,numrows,1))

SUM(OFFSET(LaneCalc!$C$1,startrow,1,numrows,1))

E5/(G5-F5)
```

Calculation time for all 5 years is now 0.673 seconds.

Dynamically calculate a small subset block for the COUNTIF.

0.14 seconds

Analysis of these calculations shows that over 80% of the time is being used by the COUNTIF formulae. We can reduce this using the fact that there will never be more than 300 origin Sites in this model, (it would be better still to count the number of unique Sites).

```
COUNTIF(OFFSET(LaneCalc!$A$1,PreviousOriginLastRow,
0,300),A7)
```

This reduces the calculation time to 0.14 seconds.

So the calculation time has been reduced from 3 hours to 0.14 seconds, which is an improvement factor of over 77000!

Links

Worksheet Links

Minimise Worksheet Links

Referencing cells on another Worksheet is nearly always slower than referencing cells on the same Worksheet. This overhead is larger when referencing formulae than when referencing data.

Make sure you have a logical flow of calculations across your worksheets.

Forward Worksheet Cross-references and **Circular Worksheet cross-references** (Sheet1 references Sheet2 references Sheet1) can be extremely slow. You really need to eliminate them:

- Rename your Worksheets to optimise the Excel's Worksheet calculation sequence.
- Move formulae between sheets.
- Merge entire sheets.

The **FastExcel** Optimise Worksheet Sequence command will find forward Worksheet cross-references, and determine a near-optimum Worksheet calculation sequence that minimises the number of forward cross-references. It will also list any circular Worksheet cross-references it finds.

Inter-Workbook Links:

Minimise Workbook links, and make sure you have a logical flow of calculations across your Workbooks.

Because Excel 97 and 2000 calculate Workbooks in the sequence of their alphabetic names, make sure that the names of linked Workbooks that you will have open at the same time sort into a sequence that gives an orderly flow of information through the links:

Avoid forward links:

- For example a Workbook named Answers linked to a Workbook named Calculations (Answers cannot be fully calculated before Calculations).

Avoid circular links between Workbooks:

- For example Book1 links to Book2, Book2 links to Book3, and Book3 links to Book1.

Excel Calculation Information

Excel's Smart Recalculation Engine

Excel normally only calculates the minimum number of cells possible.

Excel has a complex and finely tuned algorithm for choosing the fastest sequence for calculating cells and getting the correct answer.

This algorithm was changed significantly when the calculation engine was rewritten in Excel 97, and further significant changes have been made in Excel 2002. There are only minor changes in Excel 2003.

Excel's Smart Recalculation engine normally minimises calculation time by tracking changes and only recalculating:

- Cells, formulae, values or names that have changed or are flagged as needing recalculation.
- Cells dependent on other cells, formulae, names or values that need recalculation.

However defined Names are recalculated each time a formula that directly refers to the name is recalculated.

Note that the process of calculating cells that are dependent on previously calculated cells continues even if the value of a cell does not change when it is calculated.

This smart recalculation usually only takes a fraction of the time for a full calculation of all the formulae in the workbook, although sometimes the extra work required can make it slower than a full calculation. So if a recalculation takes a long time, try a full calculation to see if it is faster.

Worksheets and Workbooks are then also flagged as containing uncalculated cells.

There are exceptions to this change tracking and Smart Recalculation process:

- Opening an Excel97 workbook in Excel 2000 (or an Excel2000 book in Excel2002) in Automatic mode causes a Full Calculation rather than a smart recalculation.
- Volatile functions are always calculated.
- Full Calculation (Ctrl-Alt-F9), which calculates ALL cells.
- More than 65536 Dependencies, which causes full calculations rather than smart recalculations.
- Setting the Worksheet.EnableCalculation property to False will prevent uncalculated cells on a worksheet from being calculated. Setting Worksheet.EnableCalculation back to True will then cause **all** the formulae on the sheet to be flagged as uncalculated.
- Unused names are **never** calculated.
- Names are calculated each time they are referenced by a formula that is recalculated

Circumstances causing a formula to be evaluated.

A formula or part of a formula will be immediately evaluated whenever:

- It is entered or changed.
- It is entered or changed via the Function Wizard.
- It is entered as an argument in the Function Wizard.
- It is selected in the formula bar and F9 is pressed (press Esc to Undo and revert to the formula).

A formula will be flagged as uncalculated whenever it refers to (depends on) a cell or formula that has been

- entered.
- changed.
- is in an Autofiltered list and the criteria pulldown has been activated.
- is flagged as uncalculated.

Certain actions will also unexpectedly flag a formula as uncalculated or trigger a recalculation: see volatile actions.

A formula that is flagged as uncalculated will be evaluated whenever the Range, Worksheet, Workbook or Excel instance that contains it is calculated or recalculated.

Note that a recalculation may also occur when opening a workbook in Automatic mode or when saving a workbook in Manual Mode with the Calculate before Save option. (See Controlling Calculation)

Circumstances causing a name to be evaluated.

The circumstances causing a name to be evaluated are not the same as a formula in a cell.

- A name seems to only be evaluated when a formula that refers to it is evaluated.
- A name is evaluated every time a formula that refers to it is evaluated, so that using a name in multiple formulae can cause the name to be evaluated multiple times.
- Names that are NOT referred to by any formula are not calculated even by a full calculation (Ctrl/Alt/F9).

Excel's Calculation Process

Excel does not calculate cells in a fixed order, or by Row or Column. Instead, Excel dynamically determines and remembers its own internal calculation sequence, which is based on dependencies and dependency trees and chains.

Overall Excel has a two-step calculation process:

- Build Dependency Trees and flag cells as uncalculated. This step is executed at each cell change.
- Determine the dependency chain calculation sequence, calculate the formulae according to the calculation sequence, and remember the calculation sequence for the next recalculation.

The first step is executed at each cell entry or change. Normally this executes so fast that you do not notice it happening.

The second step is executed at each calculation or recalculation.

Note that when the value of a cell that has been calculated does not change, cells that are dependent on the calculated cell are still recalculated.

The second time you calculate a workbook is often significantly faster than the first. This is due to three factors:

- Excel usually only recalculates cells that have changed, and their dependents.
- Excel stores and reuses the most recent calculation sequence so can save most of the time used to determine the calculation sequence.
- Within an Excel session both Windows and Excel will cache recently used data and programs for faster access.

Dependencies

Dependencies control the way cells are flagged as uncalculated, and the final sequence they must be calculated in

Unreferenced dependencies in functions can give you unexpected (wrong) answers.

When a cell in a spreadsheet refers to another cell it must be finally calculated **after** the cell it refers to. This is called a **Dependency**.

Excel recognizes dependencies by looking at each formula and seeing what cells are referred to.

Understanding this is important for User Defined Functions because you need to make sure that all the cells the function uses are referred to in the function arguments. Otherwise Excel may not be able to correctly determine when the function needs to be calculated, and what its dependencies are, and you may get an unexpected answer. Specifying Application.Volatile or using Ctrl/Alt/F9 will often enable Excel to bypass this problem, but you still need to write your function to handle multiple executions per calculation cycle and uncalculated data.

See the separate page on User-Defined Functions for further explanation.

Forward References

A Forward reference refers to a cell that has not yet been calculated.

The dependency can be either a **backward dependency/backward reference** (refers to a cell that has already been finally calculated) or a **forward dependency/forward reference** (refers to a cell that has not yet been finally calculated)

These dependencies are called:

Within Sheet dependencies/references when a cell refers to a cell on the same Worksheet.

Inter Sheet dependencies/references when a cell refers to a cell on another Worksheet in the same Workbook.

Inter Workbook dependencies/references when a cell refers to a cell in another Workbook.

For example:

A1=A2+5

A2=A3+1

A3=2

These cells have to be finally calculated in the order **A3 A2 A1**.

Cell A1 **forward references** cell A2 and cell A2 **forward references** cell A3.

Cell Calculation Sequence

Excel's calculation sequence is dynamic, and may change from calculation to calculation.

Sometimes Excel will decide that the fastest way to calculate the workbook involves calculating a particular cell more than once.

Often when Excel calculates a Worksheet it discovers an improved calculation sequence, which it will then use in subsequent calculations.

Some actions such as Range Calculate can reset the initial internal calculation sequence:

Selecting a single rectangular range and doing a Range Calculate will reset the initial calculation sequence of all the formulae within the rectangular range to left-to-right, top-to-bottom. 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.

Measuring the time taken to determine the cell calculation sequence.

The time taken by Excel to work out the calculation sequence can be increased by heavy use of functions such as INDIRECT and OFFSET which increase the complexity and volatility of the dependency tree. Comparing the time taken for a Range Calculate for the used range on a worksheet with the time taken for a Sheet Calculate immediately after the Range Calculate will give you a measurement of the time taken to determine the optimum calculation sequence for a worksheet. If you do the Range Calculate followed by two Sheet Calculates a comparison of the second Sheet Calculate with the first will give you a measurement of the volatility of the formulae and the dependency tree.

Excel's usual calculation sequence is to start by evaluating the most recently entered uncalculated cells first.

Excel's default calculation sequence

The default initial calculation sequence for Excel is to evaluate cells in the reverse sequence in which formulae were entered/changed on a sheet (most recently entered first, or LIFO), and then to modify this sequence as required by the dependency trees. The final calculation sequence is determined so that all cells in dependency chains are calculated in the correct order. For most Excel formulae and functions this process of sorting the cells into dependency chain sequence is relatively fast, and the dependency chain sequence is stored after each calculation so that subsequent recalculations are even faster.

Worksheet Overhead

If the time taken by a Range.Calculate for all the formulae on a sheet is significantly less than the time taken by a Sheet.Calculate, then the organisation and volatility of the formulae on the sheet may be causing a bottleneck in determining the dependency chain sequence. This can for example be caused by large numbers of OFFSET formulae.

User Defined Functions and Dependencies

For User Defined Functions Excel has to execute the function in order to determine if it contains hidden dependencies in references to cells which are not in the argument list. Note that if you put a false dependency in the argument list (a reference which is not actually used inside the function) Excel will execute the function, but not necessarily in the sequence you expect. Also Excel will not recognise a dependency that is bypassed by an IF statement. These factors can cause problems in your UDF unless you take the appropriate precautions.

Circular Reference Cell Calculation Sequence

When your workbook contains circular references Excel has to use a slightly different calculation algorithm:

The first step is the same as a recalculation (or full calculation, depending how the calculation was invoked) that does not involve circular references. This enables all the cells that come before the circular reference to be properly calculated and the list of cells in the circular reference to be identified together with their dependents.

The second step occurs once on each iteration:

- The list of cells in the circular reference is calculated sheet by sheet, in alphabetic sheet name sequence (even in Excel 2002). Within each sheet the list of cells and array formulae in the circular reference, and all cells that are dependent on them, is calculated from left to right and top to bottom. (all the columns in the first row from left to right, then the next row ...). Each array formula in the list is calculated as a single block of cells from left to right and top to bottom, so if you want to use calculation by columns rather than by rows you have to use a multi-cell column array formula.
- This calculation ignores the dependency tree and relies on the iterative calculation process to resolve dependencies.
- Volatile cells that are not dependent on the cells in the circular list are NOT recalculated on each iteration.

This step is repeated until the iteration limits (maximum number of iterations and maximum cell change per iteration) are reached.

Microsoft's documentation of the calculation algorithm

More details of Excel's calculation algorithm and the changes made in Excel 2002 have recently been published by Microsoft on their MSDN site. The Decision Models site has a link to this document.

Workbook and Worksheet calculation sequence.

Excel97 & 2000 calculate Worksheets in alphabetic name sequence. A separate calculation sequence list is maintained for each sheet.

Excel 97 and Excel2000 calculate Workbooks, and Worksheets within each Workbook, in alphabetic name sequence:

“Book1” calculates before “Book2”.

Within a Workbook “Balance Sheet” calculates before “Cash Flow”, which calculates before “Profit Loss”, regardless of the visible order of the Worksheet Tabs in the Workbook.

A separate dependency chain calculation sequence is developed for each sheet, together with a calculation flag for each sheet and each book. Significant calculation speed gains can often be made by determining the optimum worksheet naming sequence. See the FastExcel Optimise Worksheet Sequence command.

Excel2002 also calculates worksheets in alphabetic sequence but develops a single dependency chain sequence for each workbook. For complex workbooks containing many forward worksheet references and/or circular worksheet references using Excel 2002 can significantly speed up calculation times.

Excel95 and Excel5 calculate Worksheets in the sequence in which they are defined.

(Thanks to Rob Bovey for this tip).

Dependency Trees

Dependencies are important to fast recalculation.

Excel tracks what has changed since the last recalculation, and builds **Dependency Trees** to attempt to minimise calculation time. These allow Excel to recalculate only:

- Formulae/Names that have changed.
- Formulae containing Volatile Functions
- Formulae dependent on changed or volatile formulae or cells or names.

Excel determines dependencies by looking at the cells referred to by each formula and by the argument list of each function.

Dependency trees are immediately updated whenever a formula is entered or changed.

In Excel 2002 you can force the dependency trees to be rebuilt using CTRL/ALT/SHIFT/F9.

In a complex Workbook Excel may spend significant time building and evaluating the dependency tree, and will only track a maximum of 65,536 dependencies to unique references.

When that limit is reached Excel no longer requires to use the memory needed for the dependency trees. If you have a workbook which is just short of that limit you may find that a Full Calculation is faster than a Recalculation.

Experiments indicate that dependencies are handled as follows in Excel97, Excel2000 and Excel 2002:

Within Sheet Dependency Trees:

Within Sheet dependency trees are dynamically maintained each time a Worksheet is altered.

Excel 97 and 2000 build a **dependency tree** for each changed cell/name on each Worksheet using something like the following algorithm:

Whenever a cell/name is changed:

- Flag the changed cell/name as uncalculated.
- Scan the Worksheet top-to-bottom and left-to-right looking for cells that refer to an uncalculated cell/name, and flag them as uncalculated.
- Repeat the scan as many times as necessary to resolve all dependencies.

In extreme forward referencing cases with a deep dependency tree (a long chain of formulae) this process can take a significant amount of time.

The dependency tree is updated at each change even when calculation is set to manual.

Within-Sheet forward dependencies do not usually have any significant effect on calculation/recalculation time, although this may not be true for complex volatile dependencies and recalculation.

Inter Sheet Dependencies can increase calculation times significantly.

Inter Sheet Dependency Trees:

Calculating a reference to a cell on the same worksheet is faster than calculating a reference to a cell on another worksheet.

Excel 97 and 2000 maintain a sheet-level dependency tree rather than a full dependency tree at the cell level for worksheet cross-references. Worksheets will be correctly flagged as containing uncalculated cells when they contain formulae dependent on an uncalculated cell/name on another worksheet.

Excel 97 and 2000 handle inter-sheet dependencies by calculating each sheet containing uncalculated cells in turn, in the sequence given by their alphabetic names. The calculation process for each sheet only looks at cells that are currently flagged as uncalculated. The calculation process for each sheet only looks at cells that are currently flagged as uncalculated.

If each sheet only refers backwards then the process is complete after each sheet has been calculated once, and all cells and sheets will have been flagged as calculated.

If any sheets contain cells with uncalculated **forward worksheet cross-references**, (references to uncalculated formulae on worksheets that are calculated after the current worksheet), these cells cannot yet be calculated and so the worksheet will still be flagged as containing uncalculated cells. This process continues until there are no remaining uncalculated sheets and cells.

Forward worksheet cross-references (forward inter sheet dependencies) can cause uncalculated cells to be evaluated multiple times and can increase your calculation times significantly.

When these forward worksheet cross-references become linked together into one or more circular worksheet cross-reference paths calculation time can increase even more.

You can detect this effect with **FastExcel** by looking at the Workbook Overhead: this shows the amount of time due to forward worksheet cross-references.

You can also analyse the worksheet cross-references by using the **FastExcel Optimise Worksheet Sequence** command. This will give you a recommended worksheet calculation sequence, and show you the circular worksheet cross-reference paths it has found.

Excel 2002 has a significantly improved calculation algorithm for handling forward Worksheet cross-references.

Excel2002 also calculates worksheets in alphabetic sequence but develops a single dependency chain sequence for each workbook. For complex workbooks containing many forward worksheet references and/or circular worksheet references using Excel 2002 can significantly speed up calculation times.

Inter Workbook Dependencies

These are handled similarly to inter sheet dependencies, and therefore complex forward inter workbook dependencies will increase your calculation times significantly.

Names

Names are only calculated when they are referred to, directly or indirectly, by formulae on the worksheet: unused names are not calculated even by full Calculation (Ctrl_Alt-F9).

Controlling Calculation

Excel has a range of options allowing you to control the way it calculates. You can change these options using the Tools -->Options-->Calculation tab.

Calculation Settings Keep Changing

Because a number of Excel's calculation settings work at the application level (they are the same for all open workbooks), and are set by the first workbook opened, they may appear to change randomly depending on the sequence in which workbooks are opened. FastExcel Version 2 allows you to solve many of these problems.

Automatic Calculation.

Automatic calculation mode means that Excel will automatically recalculate all open workbooks at each and every change, **and whenever you open a workbook.**

Usually when you open a workbook in Automatic mode and Excel recalculates you will not see the recalculation because nothing will have changed since the workbook was saved.

An exception is when you open a workbook in Excel 2000 that was saved using Excel 97, or you open using Excel2002 a workbook saved in Excel2000: because the Excel calculation engines are different a Full calculation is done.

Manual Calculation.

Manual calculation mode means that Excel will only recalculate all open workbooks when you request it by pressing F9 or Ctrl-Alt-F9, or when you save a workbook.

For workbooks taking more than a fraction of a second to recalculate it is usually better to set calculation to manual.

Excel tells you when the Workbook needs recalculation by showing calculate in the Status bar. If this message won't disappear see Status bar shows Calculate.

Automatic except Tables.

Excel's Data Tables feature is designed to do multiple calculations of the workbook, each driven by different values in the table. So using Automatic except Tables will stop Excel from automatically triggering the multiple calculations at each calculation, but will still calculate all dependent formulae except tables.

Application level settings.

The following calculation settings are held at the Application level rather than for each workbook, so they apply to all open workbooks.

- Calculation Mode (Automatic, Automatic except Tables, Manual, Calculate before Save).
- Iteration settings (Iteration On/Off, Max Iterations, Max Change).

Rather confusingly these application level settings are saved in each workbook, but only the settings in the first workbook opened are actioned: **the settings in subsequent workbooks are ignored.**

Workbook level settings.

The following calculation settings are held at workbook level, so each open workbook can have different settings.

Press F9 if the status bar shows "Calculate"

- Update remote references.
- Precision as displayed.
- 1904 date system.
- Save external link values.
- Accept labels in formulae.

All these workbook level calculation settings are saved and restored with the workbook, **except Update remote references**.

Controlling Excel's Calculation Options.

FastExcel Version 2 significantly extends Excel's calculation options. See the FastExcel Version 2 User Guide for details.

The details below refer to Excel's standard calculation options.

Calculation Mode usually operates at Application rather than Workbook level. FastExcel V2 allows you to control this at Workbook level.

When Excel has no workbooks open, or when you start Excel, it sets the **initial calculation mode and settings** from the **first** non-template, non-addin workbook that you open

This means that the calculation mode setting in subsequently opened workbooks will be ignored, although you can change the mode yourself at any time using Tools-->Options-->Calculation. As soon as you change the calculation mode, it applies to all subsequently opened workbooks.

If you need to override the way Excel sets the **initial calculation mode** you can set it yourself by creating a module in ThisWorkbook (doubleclick ThisWorkbook in the Project Explorer window in the VBE), and adding this code. This example sets calculation to Manual.

Note that this will NOT prevent the workbook being recalculated if it is opened in Automatic mode.

```
Private Sub Workbook_Open()
```

```
Application.Calculation = xlCalculationManual
```

```
End Sub
```

If calculation is set to Automatic when a workbook containing this code is opened, Excel will start the recalculation process before the Open event is executed. You can bypass this problem by:

- Using FastExcel version 2's Initial Calculation setting.
- Have a single workbook in your XLSTART directory which has been saved in Manual mode.
- Have an Addin installed which sets calculation to manual in its WorkBook_Open or Auto_Open procedures (Excel 97 and Excel 2000 only).

If you have a workbook that was saved in Automatic mode, but you want to open it in Manual mode:

- Open Excel with a blank workbook.
- Switch to Manual (Tools-->Options-->Calculation).
- Open the workbook that was saved in Automatic.

If you have a workbook that was saved in Manual mode, but you want to open it in Automatic Mode:

- Open Excel with a blank workbook.
- Switch to Manual (Tools-->Options-->Calculation).
- Then switch back to Automatic. The two steps are required to lock in the mode change.
- Open the workbook that was saved in Manual.

Excluding Worksheets from Recalculation.

You can prevent worksheets from being calculated or recalculated by using VBA to set the Worksheet.EnableCalculation property to False. See Calculating from VBA.

Note that this setting is NOT saved with the workbook.

FastExcel Version 2 gives you much better control over which worksheets are calculated using FastExcel MixMode settings

Recalculate before Save

In Manual mode this checkbox controls whether Excel will recalculate the Workbook as part of the save process. The default is yes.

Iteration

Usually it pays to un-check the Iteration box.

If you have intentional circular references in your Workbook, these settings allow you to control the maximum number of times the workbook will be recalculated (iterations) and the convergence criteria (maximum change: when to stop). The default should be to un-check the iteration box so that Excel does not try to solve accidental circular references.

Update Remote References.

If TRUE Automatically update any remote references (DDE Links to other programs) whenever Excel recalculates

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 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.

Save External Link Values.

If TRUE Excel saves values for links to external workbooks. I recommend keeping this option as TRUE.

Accept Labels in Formulae.

This allows Excel to try and using “natural language” labels in formulae. Because there are circumstances where this will give you unexpected or ambiguous results I recommend you do NOT use this feature.

Status Bar Shows “CALCULATE”

There are four known conditions in which the status bar will show CALCULATE:

- The Calculation Option has been set to Manual and the workbook contains uncalculated formulae. Try setting calculation to Automatic (Tools-->Options-->Calculate). Note that Excel sets the calculation mode from the first workbook opened in a session: when you open two workbooks, one saved in manual mode and one saved in automatic mode, they will both have the calculation mode of the first workbook opened.
- The Iteration Option is turned on and the workbook contains circular references. Check that turning off Iteration (Tools-->Options-->Calculation) and pressing F9 shows "Circular Reference" in the statusbar.
- You are using Excel 2000 without the SR1 update and have a user-defined function that attempts to define a name and depends on a volatile function: see MSKB Q248179
- You have hit Excel's limit for tracking dependencies.

Dependency Tree Limits

Excel only tracks 65,536 dependencies to unique references for automatic recalculation. After the workbook has passed this limit, Excel no longer attempts to recalculate only changed cells. Instead, it recalculates all cells at each calculation. This limit saves the substantial amount of memory required for the dependency trees. When this happens the status bar permanently shows “CALCULATE” for all open workbooks even after closing the main workbook.

I do not know of a way of counting the number of dependencies, so it is not easy to work out how to get below the limit, but:

Large numbers of dependencies are usually caused by many formulae referencing large ranges using functions like OFFSET, INDEX, and the Lookup. You can reduce the number of dependencies by reducing the use of these functions, and also by reducing the size of the ranges they refer to. If you reduce the number of dependencies below 65536 you have to save the workbook, close and restart Excel before Excel recognizes that the number of dependencies has fallen below 65536.

Excel 2002 allows you to do a full recalculation with dependency tree rebuild, but this does not alter the 65,536 limit.

When you have more than 65536 dependencies pressing F9 will cause a recalculation even when in Automatic mode.

Because there are other conditions that can cause "Calculate" to appear in the status bar, test for the other conditions:

- Turn off Iteration, press F9 and look for "Circular Reference" in the statusbar.
- Press Ctrl/Alt/F9 to do a full calculation. If you have hit the limit the status bar will show the calculation % climbing to and reaching 100%, and then being replaced by "Calculate"

For Microsoft's official view on the dependency limit see MSKB Article Q243495 "Calculate Message Remains in Status Bar If 65,536 Formula References"

Full Calculation and Re-Calculation.

Normally Excel calculates each cell as you enter it, and minimises calculation by only recalculating all the dependents of all the changed cells in all the open Workbooks.

You can request a **full calculation** (all formulae) by pressing Ctrl-Alt-F9, or a **recalculation** (all new/changed/volatile formulae, cells and their dependents) by pressing F9.

Usually (but not always) a **recalculation** is faster than a **full calculation**.

In Excel 2002 you can request a **full calculation with dependency tree rebuild** by pressing Ctrl-Alt-Shift-F9.

Sheet Calculation, Range Calculation, Formula and Part Formula Calculation.

Excel has finer levels of calculation than just the workbook:

Recalculate Selected Worksheet(s) (Shift-F9)

Calculate Worksheet flags the Workbook as uncalculated.

In Manual Mode:

Recalculates all uncalculated and volatile cells on the selected worksheet, or if multiple worksheets are selected they are all calculated.

Note that this only gives you “correct” results if all precedent inter-sheet and inter-workbook dependencies have already been fully calculated, and that cells on other worksheets that are dependent on these sheet(s) are not recalculated. After a sheet calculate the formulae in the sheet the sheet and the workbook are flagged as uncalculated (CALCULATE shows in the statusbar), unless there were no uncalculated or volatile cells.

In Automatic Mode:

Recalculates all uncalculated and volatile cells on the selected worksheet, or if multiple worksheets are selected they are all calculated. Then flags the workbook as uncalculated, which triggers an automatic recalculation to recalculate all volatile cells and their dependents on ALL worksheets, not just the selected sheet(s).

If there are no uncalculated dependents on other sheets or volatile cells then nothing happens in the triggered automatic calculation because the workbook was already calculated.

Calculate Selected Range (using Range Calculate in VBA)

Calculate Range resets the initial calculation sequence for the selected range, and flags the Worksheet as uncalculated when in Manual mode.

In Manual Mode:

Calculates all the cells in the selected range. Dependents and volatile cells outside the range are ignored. If one of the selected cells is part of a multi-cell array formula, then all the cells in the multi-cell array formula are calculated. **Note** that this calculation method ignores dependencies completely and calculates left-to-right and top-to-bottom, except in Excel 2002 and 2003 where dependencies with the selected range are included.

- 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.
- Conditional formats are NOT evaluated.
- After the cells have been calculated they are flagged as uncalculated.

Range.Calculate does not calculate conditional formats, and always calculates DATEVALUE using MMDDYY format.

Although Excel2000 SR1 is documented as containing a fix for Range Calculate it still works as above.

In Automatic Mode:

Range Calculate triggers an automatic recalculation but does not itself explicitly calculate the cells referred to by the range.calculate. If there are no volatile cells then no calculations are done because the workbook was already calculated.

If there are any volatile cells then ALL the volatile cells and their dependents on ALL worksheets are recalculated.

Excel 2002 Range Calculate Problems:

Range Calculate will fail in Excel 2002 if:

- Calculation is set to Manual and Iteration is enabled.
- The range being calculated contains a complete circular reference chain of formulae (you get an error message with Excel 2002 SP3)
- The range being calculated partially intersects a multi-cell array formula.
- The Range.Calculate or the Selection.Calculate Methods Fail After You Update Links to a Dynamic Data Exchange (DDE) Server. See MSKB 823338
- re-entrant calculation is not allowed: you cannot call Range.Calculate whilst a calculation is in progress
- range.calculate will fail if multiple sheets are selected

If the range you calculate includes a multi-cell array formula and you subsequently recalculate, the cells in the multi-cell array formula will be individually calculated.

The FastExcel V2 calculate range command bypasses many of these problems.

Calculating a Single Formula or Part Formula

Select the formula in the Formula Bar or highlight a part of the formula in the Formula Bar and press F9. The formula or part formula is replaced by the result.

For an **array formula** you will see an array of results, which is a wonderful way of debugging array formulae!

If you press Ctrl-Z (Undo) or Esc then the formula reappears, but if you press Enter the formula or part formula is permanently replaced.

Re-entering the = sign on the front of a formula will make Excel recalculate it in a similar way to Range.calculate, and will update the dependency tree for that formula.

Step-by Step Formula Evaluation in Excel 2002

Excel 2002 has an Evaluate Formula button that allows you to watch as Excel steps through calculating a formula in the selected cell.

Controlling Calculation from VBA.

For information on what the calculation methods do see Calculation Methods.

VBA allows you to control Calculation methods, properties and events:

Calculation Methods

F9 - Recalculate

```
Application.Calculate
```

Ctrl/Alt/F9 – Full Calculation

In Excel 2000 and 2002:

```
Application.CalculateFull
```

In Excel 97 there are two possible methods:

You can use SendKeys to send the CTRL/ALT/F9 key sequence. This can be tricky because Sendkeys just send the keystrokes into the keyboard buffer, and they are not normally processed until VBA has ended and you cannot guarantee that they will be processed by the correct window/application etc.

SendKeys With Windows 95/98/ME:

```
SendKeys "%^{F9}", True
```

- Use Sendkeys rather than Application SendKeys.
- The True argument causes VBA to wait for the calculation to proceed without interruption. This is required unless the Sendkeys statement is that last VBA statement to be executed other than End.
- The True argument does not work with Application.SendKeys

SendKeys With Windows NT/2000/XP:

```
Application.SendKeys "%^{F9}"  
DoEvents
```

If the VBA procedure containing the Sendkeys statement is called directly or indirectly from a command button the True argument does not work, so you have to use DoEvents to get Win Xp and Excel97 to process them.

Another method uses the **worksheet.enablecalculation** property. When this property is changed all the formulae on the worksheet are flagged as uncalculated, so toggling the property to false and then back to true for all sheets in all open workbooks will cause the next calculation to be a full calculation.

```
Dim oSht as worksheet  
Application.Calculation=xlCalculationManual  
  
for each oSht in Worksheets  
oSht.enablecalculation=false  
oSht.enablecalculation=true  
next oSht  
  
Application.calculate
```

You can also use this method for a single worksheet to do a **full sheet calculate**.

Full Calculation with Dependency Rebuild

In Excel 2002 only:

```
Application.CalculateFullRebuild
```

In prior versions of Excel you can achieve the same effect by switching to manual, replacing all equal signs in formulae with equal signs and then either switching back to automatic or doing a manual full calculation (Ctrl/Alt/F9)

Shift F9 – Sheet Calculate

`Worksheets("SheetName").Calculate`

Range Calculate: see Calculation Methods for details and limitations

`Range("a3:z9").Calculate`

Evaluate Method:

You can use the Evaluate method of the Application, Worksheet or Chartobject to return the result of calculating a string containing Names, Ranges and/or a Formulae to VBA without altering anything on a worksheet. Evaluate also enables you to get the results of a single-cell or multi-cell array formula passed to Evaluate as a string. There are two different syntaxes for Application.Evaluate, for example as `Evaluate("SUM(A1:A20)")` or as `[SUM(A1:A20)]`

The difference between using Application.Evaluate and Worksheet.Evaluate is not spelled out in Excel's help. Application.Evaluate the string as though it was on the active sheet, but Worksheet.evaluate evaluates the string as though it was on the referenced sheet:

If Sheet1!A1 contains 'fred' and Sheet2!A1 contains 'Joe', and Sheet 1 is the active sheet then `Evaluate("A1")` returns 'fred' but `Worksheets("Sheet2").Evaluate("A1")` returns 'Joe'

- The string must be less than 256 characters.
- Dates should be in USA format (Month-Day-Year).
- Evaluate always treats string formulae as array formulae.

If Evaluate cannot evaluate the string it returns an error rather than raising an error, so to trap the error you need to assign the result to a variant and then check the variant using ISERROR.

Properties controlling Calculation Options:

Application.Calculation

Can be set to xlCalculationAutomatic, xlCalculationManual or xlCalculationSemiAutomatic. (in Excel95 these were xlAutomatic etc)
Resetting calculation to xlCalculationAutomatic will trigger a recalculation.

Setting these properties sometimes fails with a 1004 error when the code is called from the event code of a Control Toolbox control. You can bypass this problem by setting the TakeFocusOnClick property of the control button to false, or for other controls by preceding it by Activeshsheet.Activate.

Excel's Initial Calculation Setting

FastExcel V2 allows you to control Excel's initial calculation sequence.

Excel sets the initial calculation mode from the first non-template, non-addin workbook opened, or created and calculated.

This means that the calculation mode setting in subsequently opened workbooks will be ignored.

If you need to override the way Excel initially sets the calculation mode you can set it yourself by creating a module in ThisWorkbook (doubleclick ThisWorkbook in the Project Explorer window in the VBE), and adding this code. This example sets calculation to Manual.

```
Private Sub Workbook_Open()  
Application.Calculation = xlCalculationManual  
End Sub
```

Unfortunately if calculation is set to Automatic when a workbook containing this code is opened, Excel will start the recalculation process before the Open event is executed. One way to avoid this is to open a dummy workbook with a Workbook open event which sets calculation to manual and then opens the real workbook.

Application.CalculateBeforeSave

True or False. If calculation is Manual and CalculateBeforeSave is true when you Save the workbook it will be recalculated.

Application.Iteration, MaxIterations and MaxChange

If Application.Iteration is true and the workbook contains circular references, then calculation will iterate until either the number of iterations reaches MaxIterations or the largest change in cell value in the latest iteration is less than Maxchange.

Workbook.PrecisionAsDisplayed

If True the workbook will be calculated using the number of decimal places in the formatting.

Workbook.Date1904

If true date calculations in the workbook will be based on 1904.

Workbook.AcceptLabelsInFormulas

If true Excel will use "Natural Language" labels in formulae. For anything other than simple models this should be turned off.

Preventing Worksheets being Calculated

Setting the Worksheet property `EnableCalculation` to `False` will prevent Excel from including the worksheet in a recalculation. This will stop the sheet being recalculated by `F9`, `Ctrl/Alt/F9` and by `Sheet.Calculate`, `Application.Calculate`, `Application.CalculateFull` and `Application.CalculateFullRebuild`. The `EnableCalculation` property has no effect on `Range.Calculate`.

If you use `Range.Calculate` on a formula on a worksheet that has `EnableCalculation` set to `false` then at the next recalculation Excel will recalculate any formulae on other sheets that are dependent on that formula.

Setting the property back to `true` will flag all the formulae on the worksheet as uncalculated and trigger a recalculation in automatic mode. You can use this method as a way of simulating `Calculatefull` at worksheet rather than workbook level.

The property is reset to `true` when a workbook is opened.

FastExcel Version 2 gives you an improved implementation of this facility, called Mixed Mode sheets, which allows you to control which sheets will be recalculated by which type of calculation event, and saves your settings with the workbook.

Calculation Events:

Application or Workbook SheetCalculate

This event occurs after any worksheet is recalculated or changed chart data is replotted.

Worksheet or Chart Calculate

This event occurs after the Worksheet is recalculated or the chart's changed data is replotted.

Excel 2002 Only:

Excel 2002 considerably enhances your ability to control calculation from VBA:

Adding specified cells to the calculation list

You can use `Range.Dirty` to add the specified cells to the list of cells requiring calculation at the next recalculation.

Checking Calculation Status

The `Application.CalculationState` property allows you to check if calculation has completed (`xlDone`), is pending (`xlPending`), or is in process (`xlCalculating`). The Pending state seems to correspond to the message Calculate in the statusbar: for workbooks with more than 65536 dependencies the `CalculationState` is always `xlPending` or `xlCalculating`.

Interrupting Calculation

You can control the users ability to interrupt calculation using

```
Application.CalculationInterruptKey= xlAnyKey |  
XLEscKey | xlNokey
```

Preventing Recalculation except for specified range

You can stop recalculation except for a specified range.

```
Application.CheckAbort (KeepAbort:=rngSubtotal)
```

Volatile Functions and Actions.

A Volatile function is **always recalculated at each recalculation** even if it does not appear to have any changed precedents.

Avoid Volatile functions wherever possible.

You can make a User-Defined Function volatile by including Application.Volatile in the function code.

FastExcel measures Workbook Volatility by comparing the time for a Recalculation to the time for a full calculation, and allows you to measure worksheet volatility by comparing the sheet recalculate time with the sheet full calculate time.

Excel's Volatile Functions.

Some of Excel's functions are obviously volatile: **RAND()**, **NOW()**, **TODAY()** Others are less obviously volatile: **OFFSET()**, **CELL()**, **INDIRECT()**, **INFO()** (although the **CELL("Filename")** function is not volatile)

Some are Volatile in some versions of Excel but not in others: **INDEX()** became non-volatile in Excel 97.

A number of functions that are documented by Microsoft as volatile do not actually seem to be volatile when tested:

INDEX(), ROWS(), COLUMNS(), AREAS()

You can download volatileFuncs.zip from the Decision Models website for a test workbook that shows if a function is volatile.

Volatile Actions.

Clicking Row or Column Divider

If calculation is set to Automatic clicking or doubleclicking a row or column divider will trigger a recalculation. But manually changing the height or width of a column or row will NOT trigger a recalculation.

In Manual mode these actions do not flag the workbook as requiring calculation.

Inserting or Deleting Rows, Columns or Cells

Insert or Delete Rows or Columns or Cells anywhere on a Sheet, even to the right or below the Used Range.

AND –

The Worksheet contains formulae which:

EITHER -

Contain Names referring to this Worksheet or other Worksheets

OR –

Refer to other Worksheets

OR –

Are referred to by other Worksheets

Then these formulae become flagged as uncalculated.

Adding, Changing or Deleting Defined Names

Any action taken to add, delete, change or alter a defined name or its refersto property will trigger a recalculation.

Renaming Worksheets, Changing Worksheet Position

Changing the name of a worksheet or moving it will trigger a recalculation in Automatic mode.

In Manual calculation mode, the statusbar will show "Calculate" ONLY if any formulae on any other sheet refer to the sheet with the changed name/position.

Hiding or Unhiding Rows in Excel 2003

In Excel 2003 hiding or unhiding rows will flag the selected rows as uncalculated, even if no rows were actually hidden or unhidden. If calculation is automatic this will trigger a recalculation. This behaviour is a change from previous versions. Hiding or unhiding columns does NOT flag the column as uncalculated.

Probably the reason for the change is that the SUBTOTAL function in Excel 2003 has an option to include or ignore hidden rows, so Excel needs to be able to trigger a dependency recalculation by dirtying the cells when a row is hidden or unhidden.

Formula Evaluation Circumstances

See evaluation circumstances for the circumstances that will cause a formula to be evaluated.

User-Defined Functions

User Defined Functions can provide great power and convenience and appear very simple to write. But there are some problem areas that may need special attention in your UDF coding:

- The UDF code must be in a General Module, not a Sheet or Workbook or Class module.
- Action ignored: UDF "does nothing"
- Not recalculated when needed or always recalculating.
- Unexpectedly returns #Value or other error.
- Calculates more than once in a recalculation, the Function Wizard or when entered.
- Slow to calculate.

UDF action being ignored.

Excel will not allow a UDF written in VBA to alter anything except the value of the cell in which it is entered.

You cannot make a VBA UDF which directly:

- Alters the value or formula or properties of another cell.
- Alters the formatting of the cell in which it is entered.
- Alters the environment of Excel. This includes the cursor.
- Uses FIND, SpecialCells, CurrentRegion, CurrentArray, GOTO, SELECT, PRECEDENTS etc : although you can use Range.end.
- Note you can use FIND in Excel 2002

UDF not recalculating or always recalculating or calculating in an unexpected sequence:

Dependency Sequence Problems.

Excel depends on analysis of the input arguments of a Function to determine when a Function needs to be evaluated by a recalculation. What this means is that the function will only be flagged as needing evaluation when one or more of the input arguments change (unless the function is volatile).

If your UDF gets input values from any cells that are not in its argument list then it may not be recalculated, and give the wrong answer. Mostly you can bypass this problem by doing a Full Calculation (Ctrl-Alt-F9), rather than a recalculation (F9), or by making your UDF volatile, but I strongly recommend that you put all the input cells in the argument list.

During a recalculation if Excel does evaluate the UDF it determines which cell references are actually being used inside the function to affect the function result, and if those cells have not yet been finally calculated it will reschedule the Function for later calculation. This is required to make the UDF be finally calculated in the correct dependency sequence.

```
Function mySum1(SheetName as string) as Variant  
mySum1=Worksheets(Sheetname) .Range("A1")+worksheets("D  
ATA") .Range("A2")  
End Function
```

This function will not automatically recalculate if Data!A2 changes: it will only automatically recalculate when Sheetname changes.

Note that putting all the references in the argument list does not control the initial sequence which Excel uses to calculate the UDF, and it may well be calculated more than once per recalculation cycle.

See UDFs evaluated more than once per workbook calculation.

False Dependencies

If you put a false dependency in the argument list (a reference which is not actually used inside the function) Excel will execute the function when the dependency changes, but not necessarily

To ensure that the code in a Function is executed in dependency sequence you need to use ISEMPTY to check the input argument(s). See Detecting Uncalculated Cells below. in dependency sequence.

User-Defined Volatile Functions

Making UDFs Volatile is NOT a good substitute for including ALL it's inputs in the argument list.

If you develop a User Defined Function (UDF) you may need to make it Volatile (Application.Volatile(True)) to make it recalculate BUT:

- Application.Volatile makes your function ALWAYS recalculate each time Excel calculates, which will slow down calculation.
- Application.Volatile does not directly affect Calculation Sequence.
- If your UDF refers to cells that are not included in the function's argument list and (if the function is calculated) those cells have not yet been calculated, then Excel will reschedule the UDF to be recalculated again later. Note that uncalculated cells do not stop your UDF from calculating, they just make it calculate more than once. This will mostly give you the correct answer.

To make your function volatile put Application.Volatile(True) in your function before any other executable statements.

```
Function mySum2(SheetName as string) as Variant
Application.Volatile(True)
mySum2=Worksheets(Sheetname).Range("A1")+worksheets("DATA").
Range("A2")
end function
```

MySum will now give you the answer you expect.

I recommend avoiding the use of Application.Volatile if at all possible: put ALL your references in the argument list

```
Function mySum3(theFirstCell as range,theSecondCell as
range) as Variant
mySum3=thefirstCell.Value+theSecondCell.value
End Function
```

Modifying UDFs and Sheet.calculate

Whenever you modify the code in a UDF Excel flags all instances of the UDF in all open workbooks as being uncalculated. If you then do a sheet.calculate or shift/F9 Excel will not only calculate the currently selected worksheet(s), but will also evaluate all the instances of the UDF that are not on the currently selected worksheets.

Unexpectedly returning #Value or being recalculated more than once.

If you are using Excel 97 make sure you have installed the SR1 and SR2 updates.

There are several problems with UDFs in Excel97 which are fixed by these two service releases, which are available from:
The Microsoft Office download centre.

UDF error with multi-area range argument.

Howard Kaikow <http://www.standards.com> has discovered a bug in Excel's processing of UDFs using non-contiguous multi-area ranges as input arguments. See AreasBugBypass2.zip for a download containing two workbooks that illustrates the problem and a way of bypassing it..

The problem occurs when:

- A UDF has a multi-area range as one of its input arguments.
- And the multi-area range refers to the worksheet that contains the formula with the UDF.
- And a different sheet is the activesheet when the UDF is calculated.

In these circumstances Excel/VBA incorrectly treats the multi-area range as referring to the active sheet. This means that the UDF may give incorrect answers without warning.

I recommend that you do not attempt to program a UDF to handle multi-area ranges as arguments: use multiple (optional if there are a varying number) range arguments instead.

If required you can bypass the problem as demonstrated in the download, or by ensuring that there are no instances of the UDF where the multi-area range refers to the sheet with the UDF formula.

UDFs may be evaluated more than once per calculation

Various conditions may cause Excel to evaluate your UDF more than once during a recalculation. This means that your UDF should:

- Explicitly initialize all variables used in the UDF.
- Error-handle input from uncalculated cells (check for ISEMPTY, unexpected zeros, blanks, missing properties etc)
- Avoid doing intensive calculations until all input cells have been fully calculated
- Cache input values and output values to avoid time-intensive calculations when the input values have not changed.

Untrapped Errors

You should make sure that you have an On Error handler in your UDF to handle both real errors and errors that are caused by out of sequence execution. Untrapped errors can halt the calculation process before it is complete.

```
Function mySum4(theFirstCell as range,theSecondCell as
range) as Variant
On error goto FuncFail:
mySum4=thefirstCell.Value+theSecondCell.value
Exit Function
FuncFail:
mySum4=CvErr(xlErrValue)
End Function
```

Unhandled UDF Errors, and debugging your UDF's

If you don't have an On Error handler in your UDF you may need to be aware of the differences in the way that different Excel versions react.

Excel97

Excel 97 (SR2 and previous) will **interrupt calculation** and return #Value for UDF's if:

- Calculation is called from VBA using Sheet.Calculate, Application.Calculate, or SendKeys "%^{F9}", True, and the UDF contains an unhandled error

If Application.Interactive=False Excel will hang, and you will need to shut it down.

Range.Calculate with Excel 97 will automatically go into debug mode highlighting the error.

If you start the calculation process using Shift-F9 or F9 this problem does not occur: all cells are calculated and debug mode is not entered.

For the Microsoft view on the Excel 97 error handling problem see MSKB 244466

Excel2000

Range.Calculate goes into debug mode when the UDF has an unhandled error.

Sheet.Calculate, Application.Calculate and Application.CalculateFull: does not usually interrupt calculation or enter Debug mode.

Excel 2002

Range.calculate, Sheet.Calculate, Application.Calculate and Application.CalculateFull: does not usually interrupt calculation or enter Debug mode.

Detecting Uncalculated Cells.

Using the Visual Basic ISEMPTY function on a UDF Range argument will return TRUE if either the input cell is not yet calculated or it contains nothing.

To distinguish between uncalculated cells and cells without any data check that the length of the formula is greater than 0 (suggested by David Cuin):

```
ISEMPTY(Cell.Value) AND Len(Cell.formula)>0
```

This expression will only return true if the cell is both uncalculated and contains a formula. Or you may decide that it is safe to skip calculation only if ALL the input cells are uncalculated. See the example below in UDF Performance.

I recommend that ALL UDFs should include both an On Error handler and where possible an ISEMPTY check. UDFs written using the C API can check for xlCoerce returning xlretUncalcd.

In this example the Function will not return a recalculated answer until both the input arguments have been recalculated and are not empty.

```
Function mySum5(theFirstCell as range,theSecondCell as
range) as Variant
On error goto FuncFail:
If IsEmpty(theFirstCell) or IsEmpty(theSecondCell) then
Exit Function
else
mySum5=thefirstCell.Value+theSecondCell.value
endif
Exit Function
FuncFail:
mySum5=CvErr(xlErrValue)
End Function
```

If the input arguments are multi-cell ranges one or more of the input cells may not contain any data.

Controlling calculation sequence using false dependencies

If you have a function that you want to calculate in dependency sequence for an argument that does not affect the result of the function (false dependency), you also need to check the input arguments with ISEMPTY.

In this example the second argument is a false dependency because it does not affect the result of the function.

```
Function mySumTimes2(theFirstCell as range,theSecondCell as
range) as Variant
On error goto FuncFail:
If IsEmpty(theFirstCell) or IsEmpty(theSecondCell) then
Exit Function
else
mySumTimes2=thefirstCell.Value*2
endif
Exit Function
FuncFail:
mySumTimes2=CvErr(xlErrValue)
End Function
```

Referencing cell formatting properties

If your UDF references cell properties other than .value or .formula (ie .Bold) there are some occasions when these properties may be undefined when your UDF is evaluated. One such occasion is renaming a worksheet in automatic mode. If this happens you may need to explicitly recalculate your function.

UDF Performance

For optimum performance UDFs should be coded in C and use the C API.

Usually Excel's built-in functions are faster than the equivalent function written in VBA, unless the VBA function uses a better algorithm.

Make sure you use an IsEmpty routine at the top of your UDF to check for uncalculated cells so that you can avoid unnecessary calculations of your UDF. IsEmpty returns True if the variable being checked has not been initialised or has been set to empty. When used on UDF input range arguments IsEmpty returns True if the range either contains nothing or has not yet been calculated.

Combining an ISEMPTY test with a test that the formula length is >0 will detect only uncalculated cells but not cells that are empty because they contain nothing.

Consider if it makes sense only to abandon calculation not only if ALL the input arguments are uncalculated/empty but also if ANY of them are uncalculated/empty.

Also since executing UDF's is usually slower than other Excel calculations, try to put the reference to your UDF in a place in your formulae where it will be calculated as late as possible (towards the end of the formulae and inside the brackets).

Automatic and Function key Calculation slower than VBA calculation

UDFs calculate significantly slower when the calculation is started automatically by Excel or by pressing F9 than when the calculation is started by a vba calculation statement like Application.calculate.

The slowdown is significantly larger if the VBE is open and not minimised.

The slowdown is an overhead for each UDF that is recalculated, so its roughly proportional to the number of UDFs.

These timings are for 16000 very simple UDFs, using Excel2002 on an AMD 1200MHZ with Windows XP:

- Autocalc with VBE open and maximised: 91 seconds
- Autocalc with VBE open and minimised: 38 seconds
- Autocalc with VBE closed: 2 seconds
- Application.Calculatefull with VBE open and maximised: 0.302 seconds
- Application.Calculatefull with VBE closed: 0.293 seconds

So if you are using a lot of UDFs it really pays to be in manual calculation mode and have a calculate button that uses VBA to initiate an Excel calculation (Application.Calculate or Application.Calculatefull).

Array Functions

UDF's can be written as multicell array formulae that can be entered using Ctrl-Shift-Enter. The results of the array calculation are returned to the cells by assigning an array to the function.

Note that Excel behaves unexpectedly when a multi-cell UDF is entered or modified and depends on volatile formulae: **the UDF is evaluated once for each cell it occupies**. This does not happen when the UDF is recalculated, only when it is entered or changed.

Transferring information from Excel Ranges to the UDF.

You should also try to minimise the performance overhead of transferring information from Excel to VBA and back to Excel:

If you are going to process each cell in the input range(s) then they should usually be read into a variant variable containing an array, which is subsequently read from for the calculations. This avoids reading information from Excel cell by cell, which is slow, and ensures that you only read each cell once. Note that this example assumes that each range is a single-area contiguous range.

```
Function mySum6(theFirstRange As Range, theSecondRange As
Range) As Variant
Dim dblMySum As Double
Dim varRange1 As Variant
Dim varRange2 As Variant
Dim blEmptyCells As Boolean
Dim j As Long
Dim k As Long

On Error GoTo FuncFail:
'
' initialise output
'
dblMySum = 0#
'
' get ranges into variant variables holding array
'
varRange1 = theFirstRange
varRange2 = theSecondRange
'
' check for non-empty cell
'
blEmptyCells = True
For j = 1 To UBound(varRange1, 1)
If Not blEmptyCells Then Exit For
For k = 1 To UBound(varRange1, 2)
If Not IsEmpty(varRange1(j, k)) Then
blEmptyCells = False
Exit For
End If
Next k
Next j
If blEmptyCells Then
For j = 1 To UBound(varRange2, 1)
If Not blEmptyCells Then Exit For
For k = 1 To UBound(varRange2, 2)
If Not IsEmpty(varRange2(j, k)) Then
blEmptyCells = False
Exit For
End If
Next k
Next j
End If
' exit function if ALL input cells are empty
If blEmptyCells Then
Exit Function
Else
' add cells to double (error traps text cells)
For j = 1 To UBound(varRange1, 1)
For k = 1 To UBound(varRange1, 2)
dblMySum = dblMySum + varRange1(j, k)
Next k
Next j
For j = 1 To UBound(varRange2, 1)
For k = 1 To UBound(varRange2, 2)
```

```
dblMySum = dblMySum + varRange2(j, k)
Next k
Next j
End If
' assign value to function
mySum6 = dblMySum
Exit Function
FuncFail:
mySum6 = CVErr(xlErrValue)
End Function
```

Using Excel functions inside your UDF.

If you are going to process the input ranges using Excel functions called from VBA, then keep the input ranges as Range object variables, so that the function does not have to transfer all the cell information to VBA : you are then just manipulating the object variable pointers.

This example extends the isempty check to disjoint ranges containing multiple areas, and will only check cells that are within the used range (efficient handling of ranges specified as entire columns or rows).

Note that because there is a bug in the way Excel handles ranges containing multiple areas I do not recommend that UDF's are programmed to handle multiple areas.

```
Function mySum7(theFirstRange As Range, theSecondRange As
Range) As Variant
Dim blEmptyCells As Boolean
Dim rngUsedCellsInRange As Range
Dim oCell As Range
Dim j As Long
On Error GoTo FuncFail:
    check for non-empty cell
blEmptyCells = True
Set rngUsedCellsInRange = Intersect(theFirstRange,
theFirstRange.Parent.UsedRange)
If Not rngUsedCellsInRange Is Nothing Then
For Each oCell In rngUsedCellsInRange
If Not IsEmpty(oCell) Then
blEmptyCells = False
Exit For
End If
Next oCell
End If
If blEmptyCells Then
Set rngUsedCellsInRange = Intersect(theSecondRange,
theSecondRange.Parent.UsedRange)
If Not rngUsedCellsInRange Is Nothing Then
For Each oCell In rngUsedCellsInRange
If Not IsEmpty(oCell) Then
blEmptyCells = False
Exit For
End If
Next oCell
End If
Set oCell = Nothing
Set rngUsedCellsInRange = Nothing
    exit function if all input cells are empty
If blEmptyCells Then
Exit Function
Else
mySum7 = Application.Sum(theFirstRange, theSecondRange)
End If
Exit Function
FuncFail:
Set oCell = Nothing
Set rngUsedCellsInRange = Nothing
mySum7 = CVErr(xlErrValue)
End Function
```

UDFs with a variable number of input arguments.

This example shows how to handle a variable number of arguments.

```
Function mySum8(ParamArray varArgs() As Variant) As Variant
Dim varArg As Variant
Dim rngCell As Range
Dim dblMySum As Double
Dim blEmptyCells As Boolean
Dim j As Long
Dim k As Long
On Error GoTo FuncFail:
' initialise output
'dblMySum = 0#
' check for non-empty cell
'blEmptyCells = True
For Each varArg In varArgs
If Not blEmptyCells Then Exit For
If Not IsMissing(varArg) Then
' if not a range skip the check
If TypeName(varArg) = "Range" Then
For Each rngCell In varArg
If Not blEmptyCells Then Exit For
If Not IsEmpty(rngCell) Then
blEmptyCells = False
Exit For
End If
Next rngCell
Else
If Not IsEmpty(varArg) Then blEmptyCells = False
End If
End If
Next varArg
' exit function if all input cells are empty
If blEmptyCells Then
Exit Function
Else
' add cells to double (error traps text cells)
'For Each varArg In varArgs
If Not IsMissing(varArg) Then
If TypeName(varArg) = "Range" Then
For Each rngCell In varArg
dblMySum = dblMySum + rngCell
Next rngCell
Else
dblMySum = dblMySum + varArg
End If
End If
Next varArg
End If
' assign value to function
'mySum8 = dblMySum
Exit Function
FuncFail:
mySum8 = CVErr(xlErrValue)
End Function
```

Function Wizard

Your end-users can use the Function Wizard to enter your UDFs (usually from the User Defined Category) into a worksheet.

If your UDF takes a long time to execute you will soon discover that the Function wizard executes your UDF several times.

There are two methods you can use to avoid this:

- Declare the function as Private: the UDF will still work but will not appear in the list of functions shown by the function wizard (but you can still use the Function Wizard to modify an existing entry).
- Add code to your UDF to determine if it has been called from the function Wizard.

Axel König has suggested adding the following code:

```
If (Not  
Application.CommandBars("Standard").Controls(1).Enabled)  
Then Exit Function
```

This code depends on the fact that when using the function wizard most icons in the toolbars are disabled. I have tested Axel's solution in Excel 97, Excel 2000, Excel2002 and Excel 2003, and it works well.

A solution is also possible by using the Windows API to check if the Function Wizard window is showing and has the same process ID as the current Excel process.

Repetitive Calculation Features and Add-Ins.

Some Excel features and Add-Ins cause Excel to repeatedly recalculate:

- Excel Tables
- Scenarios
- Circular References with iteration enabled
- Solver
- Goal Seek
- Risk Analysis Add-ins

Make sure when using these features that you really need them, and that you have optimised fully the blocks of formulae that will be repeatedly recalculated.

Circular References

Excel's ability to iteratively solve circular reference problems can be useful, but it is easy to over-use.

The default calculation mode for Excel is to disable iteration, so that if you create a circular reference and calculate, Excel detects it and warns you that you have a circular reference. To iteratively solve the circular reference you must turn on iteration (Tools-->Options-->Calculation). Once you have turned iteration ON you will not get any more messages about circular references, and Excel will repeatedly recalculate until it reaches the limit on the number of iterations, or the largest change in value during an iteration is less than the specified max change value.

Note that Excel uses a different calculation process to solve workbooks with circular references. When designing circular reference solutions you may need to take account of the calculation sequence that Excel uses for circular references.

There is one major problem with using circular references: once you have one circular reference it is very difficult to distinguish between mistakes that have created inadvertent circular references and the intentional circular reference. In most circumstances (particularly with financial calculations) it is possible (and desirable) to 'unroll' the circular reference using an extra step. For example, if you want to calculate a cash balance which includes interest on the balance you can use a circular reference where the interest depends on the balance and the balance depends on the interest. This calculates compound interest. You can 'unroll' the calculation by calculating the balance before interest, then calculating the interest (simple or compound) and then the final balance.

Intentional Circular References

If for some reason you cannot 'unroll' the calculation then I would advise using Stephen Bullen's technique of including an IF statement in your intentional circular references that acts as a switch to turn off the circular references, and sets up the initial condition for the iterative solution.

```
A1:=1  
B1:=IF(A1<>0,C1,A1)  
C1:=0.1*E1  
D1:=100  
E1:=D1+B1
```

If A1 is zero and iteration is disabled then Excel will not regard this calculation as a circular reference, so any circular reference detected is probably intentional. Set A1 to 1 and enable iteration to request Excel to solve using iteration.

Note that not all circular calculations converge to a stable solution. Another useful piece of advice from Stephen Bullen is to test the calculation in manual calculation with the number of iterations set to 1. Pressing F9 will single-step the calculation so you can watch the behaviour and see if you have genuinely converged to the correct solution.

There are a number of examples of using intentional circular references on Stephen Bullen's website.

Circular References and Calculation Speed.

Because Excel calculates circular references sheet-by-sheet without considering dependencies you will tend to get very slow calculation if your circular references span more than one worksheet. Try to move the circular calculations onto a single worksheet or optimise the worksheet calculation sequence to avoid unnecessary calculations.

Before the iterative calculations start Excel has to recalculate the workbook to identify all the circular references and their dependents. This process is equivalent to 2 or 3 iterations of the calculation.

After the circular references and their dependents have been identified each iteration requires Excel to calculate not only all the cells in the circular reference, but also any cells which are dependent on the cells in the circular reference chain. So if you have a heavy calculation which is dependent on cells in the circular reference it can be faster to isolate this into a separate closed workbook and open it for recalculation after the circular calculation has converged.

And of course it is important to minimise both the number of cells in the circular calculation and the calculation time taken by these cells.

Excel Memory

Windows Memory and RAM

Your Windows PC has various kinds of memory: physical memory (RAM), Windows virtual memory, Excel useable memory etc.

Ultimately memory is limited by the size of your swap file on your hard disk. However hard disks are very slow compared to physical memory (RAM). So Windows tries to maintain the things it needs to use in RAM and swaps stuff it doesn't need right now to disk.

The Excel program itself usually occupies between 8MB and 15 MB of memory. In addition Excel uses additional memory to store Workbook formulae and data etc.

Excel uses its own pools of memory (Excel useable memory) for this Workbook data. Unfortunately these memory pools are not in general as large as the Windows memory, and may well be a lot less than the RAM on your PC. Excel 95, Excel 97 and Excel 2000 have a limit for workbook formula memory of about 80 MB, and Excel 2002 has a limit of about 160MB.

Excel 2003 has a significantly larger memory capacity of about 1 Gigabyte (1024MB). See Memory Limits for details.

How much RAM do you need?

Paging to your virtual memory swap-file is extremely slow. You need enough physical RAM for the operating system, Excel and your workbook(s), and RAM is not expensive.

- If you have more than very occasional hard disk activity during calculation, you need more RAM.

Windows 95/98

You need about 24MB for the operating system and 8-24MB for Excel so:

- 32 MB RAM is OK for small workbooks.
- 64 MB RAM is OK unless you need to open more than about 32MB of workbooks.
- 128 MB RAM works well except when using workbooks containing large amounts of data.
- You will continue to see small speed improvements with 256MB RAM when using large workbooks, particularly with Excel 2002.

Windows ME and Windows NT

You need about 32MB for the operating system and 8-24MB for Excel so:

- 48 MB RAM is OK for small workbooks.
- 72 MB RAM is OK unless you need to open more than about 32MB of workbooks.
- 128 MB RAM is OK except when using workbooks containing large amounts of data.
- You will continue to see small speed improvements with 256MB RAM when using large workbooks, particularly with Excel 2002.

Windows 2000

You need about 64MB for the operating system and 8-24MB for Excel so:

- 72 MB RAM is OK for small workbooks.
- 128 MB RAM is OK unless you need to open more than about 32MB of workbooks.
- 196 MB RAM is OK except when using workbooks containing large amounts of data.
- You will continue to see small speed improvements with 256MB RAM when using large workbooks, particularly with Excel 2002.

Windows XP

You need about 128MB for the operating system and 8-24MB for Excel so:

- 128 MB RAM is OK for small workbooks.
- 196 MB RAM is OK unless you need to open more than about 32MB of workbooks.
- 256 MB RAM works well except when using workbooks containing large amounts of data.
- You will continue to see small speed improvements with 512MB RAM when using large workbooks, particularly with Excel 2002.
- Using Excel 2003 with very large workbooks you can make effective use of at least 1GB of RAM.

How large a Swapfile do you need?

Usually you can let Windows automatically control the size of your swapfile. If the hard disk containing your Windows system (usually C drive) has very little space you may need to manually control the size and location of your swapfile using the Windows Control Panel (System Properties, Advanced Tab, Performance Settings, Memory Usage).

If you work with large workbooks your swapfile should be at least 256MB and preferably 512MB to 1024MB.

You can reduce the amount of swapfile being used by closing programs that are loaded but not in use, and by reducing the number of programs that are loaded at startup.

Measuring Memory Used by Excel

If you are using Windows NT or Windows 2000 you can use Task Manager to track the total amount of memory being used by EXCEL.EXE.

This is the total of the memory used by Excel itself and the memory used by Excel to store data and formulae etc.

Task Manager will also show you the size and usage of your Swapfile.

You can track the amount of memory Excel is using for Workbooks etc using:

- The **FastExcel** Memory Used, Pivot Cache Memory Used and Profile Workbook commands.
- Excel's Worksheet function `INFO("memused")`
- `Application.MemoryUsed` from VBA.

There are two other memory used commands available:

- `INFO("memavail")` or `Application.MemoryFree`
- `INFO("totmem")` or `Application.MemoryTotal`

These are supposed to give you the amount of memory available or free, and the sum of memory used and memory available.

Unfortunately memory available or free does not work properly and always shows a constant 1 Megabyte available.

Consequently total memory, which shows the sum of memory used and memory free, also does not work properly.

Measuring System Memory

You can use the `GlobalMemoryStatus` windows API to obtain information about system memory status, including total and used Physical RAM, Swapfile and Virtual Memory.

See MSKB article Q213267 for details and an example of this API.

Out of Memory, Memory Limits

If you work with large Workbooks, external data or charts sooner or later you will get the dreaded "Out of Memory" or "Not enough System Resources/Memory to Display Completely" message, regardless of how much RAM or how big a swap-file you have.

This is because Excel has its own memory manager and its own memory limits.

Note that Excel 2003 will allow you to make effective use of much more RAM than previous versions.

Excel's memory manager in Excel 95, 97, 2000 and 2002 does not make use of all available RAM or virtual memory, regardless of the Operating System being used (Win95/Win98/WinMe/Win NT4/Win 2000/Win XP).

I have not experimented with Excel on the Mac so don't know if the same limits apply.

So buying more RAM or increasing your swap file size probably won't help! (see Memory Needed for recommendations on RAM and swapfile).

You may also encounter "Memory Leaks", where certain tasks such as printing or inserting graphics causes the memory Excel uses to increase, so that you eventually run out of memory and have to close Excel.

Excel has a number of memory limits which apply to different items. These limits operate relatively independently of each other. They mainly apply at the Application level, and so are cumulative across all the open workbooks, although you can open two instances of Excel and each instance will have these limits independently of the other.

This page contains a number of references like "see MSKB 313275 " which refer to articles in the Microsoft Knowledge Base (MSKB).

You can access MSKB articles using a URL of <http://support.microsoft.com/?kbid=n> where n is the identification number of the article (313275 in the example above).

The published Memory limits

Excel 5 was limited to about 16 MB of memory for Workbooks etc.(heap space), and to a maximum of about 37120 rows containing information. This was documented in MSKB [Q99345](#), which is no longer available in the online MSKB

Excel 95, Excel 97 and Excel 2000 are officially limited to 64MB of formula memory for workbooks (heap space).

Excel 2002 is officially limited to 128MB of formula memory for workbooks (heap space).

Excel 2003 is officially limited to 1 Gigabyte (GB) of formula memory for workbooks (heap space): I have successfully used over 750MB of workbook memory on a 768MB RAM system.

Note this limit excludes the memory used by the Excel program itself and the Operating system.

In Windows NT/2000/XP you can use Windows Task Manager to see the amount of memory used by the Excel process, which includes both the Excel program and the memory used for workbooks.

This limit is documented in MSKB 313275, which describes "heap space" as being memory used to:

- Track cells and formulas.
- Provide copy and paste functionality.
- Track pointers to objects.

You can find out how much "heap space" memory Excel is using with:

- Excel's worksheet function INFO("memused")
- Application.MemoryUsed from VBA.

Track cells sounds as though it refers to cells containing data, but this is misleading. This limit excludes cells containing data. In practice you can usually use between 64MB and 80MB of formula memory with Excel 95, 97 and 2000, about 160MB with Excel 2002 and at least 1 Gigabyte (1024MB) with Excel 2003 before hitting this limit.

The following list of limits and problems is far from complete.

See also **Memory Leaks**, and **Measuring Memory**.

Workbook Memory limits:

Workbook memory is the memory used by Excel to store information about open workbooks

- Formulae, Pivot Tables and UNDO: about 80MB. **This has been increased to approx 160MB in Excel 2002, and substantially increased in Excel 2003 (probably > 1 gigabyte).**
- Maximum number of rows containing information: just over 1000000 (1 million) under Windows XP, unknown under Windows 98.
- Cells containing data: about 1GB memory under windows XP, unknown under Windows 98.
- Workbooks with both data and formulae can contain for instance 50MB of formulae and 150MB of data (provided under Windows XP that the number of rows containing information is less than about 1 million).
- Workbook file size is usually less than workbook memory, so a workbook file can be between 3MB and 60MB and still hit the 80MB memory limit.
- Opening 8 workbooks of 10MB each is the same as opening one workbook of 80MB.
- Opening a workbook may require more memory than saving it, so sometimes you cannot reopen a workbook you previously created!

Because the limits are more or less independent a Workbook containing both data and formulae can use more than 80 MB.

80MB sounds like a very large amount of formulae, but actually it represents a maximum of less than 2 million formulae, which is only about 12% of a single Worksheet.

As an extreme case I created a test workbook with only 40000 formulae whose filesize is just over 3MB, but which exceeds the 80MB formula limit when saved and reopened. The workbook contains 200 worksheets, each containing 200 formulae, and the formulae link each sheet to every other sheet.

Workbook or Excel will not open, and gives "Out of Memory" message.

If your workbook gives you an "Out of Memory" message and refuses to open, or you cannot open Excel even with a blank workbook, try these options:

1. Reboot your PC and start Excel in "Safe Mode", then try opening the workbook: click Run on the Start menu. Type the path to excel.exe, and add /s. Then press OK. Start by trying Excel /s and if that does not work you will need the full path, for example C:\Program Files\Microsoft Office\Office\Excel.exe /s Then open the workbook whilst holding the shift key down to stop any macros from executing.
2. Try using Excel 2002 to open the workbook whilst holding the shift key down, .
3. Delete all the files in your temp directory, but make sure that you have completed any pending software installations first (you may need to reboot your PC). If you are not sure where your temp directory is you can access it using Start-->Run-->%temp%-->OK. Temporary files may accumulate in your \Windows\Temp directory (Win95/98/ME), or your \Documents and Settings\<user>\local settings\temp directory (Win2K/WinXP). Too many temporary files can cause problems, so clean them out from time to time.
4. Find out where your Excel toolbar file is located using Windows search (search for *.XLB), and rename it to something like Toolbar.Old. The .XLB file is used to store your customisations of the Excel toolbars, and sometimes it gets corrupted. If Excel cannot find the .XLB file it will automatically create a new one.

For more ideas see MSKB 280504 " How to Troubleshoot Startup Problems in Microsoft Excel"

Links, Graphics, Zoom, Charts, and Fonts.

- External links: see MSKB articles 167079 - 16375 unique cells per Worksheet in a closed Workbook. This article actually slightly overstates the restriction: the limit is 16375 unique rows per worksheet rather than cells, so you can link to several columns in the same 16375 rows of a worksheet in a closed workbook.
- External Links: see MSKB 178086 and 214342 - Save External Link Values with 16000 cells in a single reference.
- Graphic objects seem to cause problems frequently.
- Using Zoom (i.e. Zoom NOT at 100%) sometimes uses a lot of memory.
- Using Zoom with Controls – see MSKB 183503
- Opening a workbook containing Charts - 172948 or 214683
- Adding Charts - 168650
- Copying a worksheet containing Charts - 264986
- Number of different Fonts exceeds 255 (point sizes count as fonts) - 255622
- Automatic Font Scaling in Charts – see MSKB 292263

Miscellaneous Memory Limits

- Capability limits of Excel97 - see MSKB Q296053
- Disk is Full message - see MSKB Q21425
- Large autofills (limit is 32760 source cells) - see MSKB Q313275
- Repeatedly calculating custom functions - see MSKB Q265023
- Pivot Table limits - see MSKB Q104308 and Q162476 for XL97 and Q211517 and Q129160 for XL2000 and Q291061 for Excel2002
- Autoformatting an entire worksheet - see MSKB Q211478
- Resetting the UsedRange in a macro - see MSKB Q244435
- Array Formula limits in XL2000 and Excel2002: max 65472 array formulas on a sheet that refer to other sheets - see MSKB Q166342
- Array Limits for various Excel versions: - see MSKB Q177991
- Too many modules and forms open in the Visual Basic editor (close them!).
- Using a macro that was recorded to open a Text file - see MSKB Q134826
- Repeatedly Hyperlinking between your Web Browser and Microsoft office - see MSKB Q157763 or Q199337 or Q297891
- Creating multiple Maps - see MSKB Q214380
- Office Clipboard maximum is 4MB or 8MB- see MSKB Q221461 and MSKB Q290373
- Unable to start Excel2000 - see MSKB Q305498
- Too many manual pagebreaks, limit is 1026 - see MSKB Q284916
- Names nested more than 20 deep - see MSKB Q292471
- Rotated Text in Embedded Workbook using NT4 - see MSKB Q169718
- Pasting Symbols in Charts with Win95/98 - see MSKB Q247578
- Incorrect use of Cells property - see MSKB Q173182 or Q213682 or Q291067
- Saving a workbook from the VBE in duak 97 7 5.0/95 format - see MSKB Q297024
- Repeatedly opening and closing workbooks containing controls - see MSKB Q238570 and Q248180
- Not enough stack space to run macro - see MSKB 111867
- Office ClipBoard is full - see MSKB 221461

For additional information search the Microsoft Knowledge Base for “not enough memory” or "Out of memory".

Reducing memory Used

You can reduce the amount of memory used by:

- Checking that the Excel's last cell on each sheet is in the right place: Ctrl-End or Edit-->Goto-->Special-->Last cell. When its not where you want, for instance because of additional formatting or recently deleted cells, you can reset the last cell by deleteing all columns to the right and rows below the correct last cell, and then saving the workbook. Note that since this process will alter any formulae that refer to the deleted cells you should make a backup copy first!
- Converting formulae to values (Paste Special Values).
- Switching off Change Tracking for Shared workbooks.
- Reducing the number of formulae referencing other worksheets. Formulae which refer to cells or ranges on other worksheets use significantly more memory for dependency trees than formulae which refer to cells or ranges on their own worksheet. The best way to do this is by moving the blocks of cells which are being referenced onto the sheet that refers to them.
- Using smaller ranges in formulae such as INDEX and VLOOKUP.
- Avoiding references to ranges containing unused or blank cells on other worksheets.
- Removing zero-sized objects.
- Using multi-cell array formulae.
- Storing repetitively used formulae in defined Names and using the Names in place of the formulae.
- Shortening the formula.
- Move the Pivot Tables to a separate Workbook.
- Reducing the number of fonts.
- Simplifying formatting.
- Reducing the number of graphic or embedded objects.
- Closing open Modules and Forms in the VBE.

Memory Leaks

Memory Leaks occur when a program (Excel, VBA...) obtains some memory but fails to hand it back to the memory manager.

Usually you only notice this if you monitor memory usage closely, or you run out of memory!

Most of the time you can reclaim the memory by saving your Workbook, closing and then restarting Excel. Sometimes you have to restart Windows.

Sometimes you can delay the point at which you run out of memory by increasing the size of your Swapfile.

You can easily monitor memory leaks with Windows NT, 2000 and XP using the Windows Task Manager.

Here is a partial list of things that can cause memory leaks:

- Page Setup in Win95 & Win98 & Win ME (GDI problem) - see MSKB Q192869
- Printing using HP Printers - see MSKB Q165985
- Replacing external link formulae
- Some external data retrieval actions, particularly repeatedly reading text files. Excel seems to store only unique strings rather than multiple occurrences of the same string, but does not seem to remove a unique string when it is no longer in use.
- Controls embedded on worksheets - see MSKB Q238570
- User Defined Functions, particularly when they take input parameters from another sheet and return a string.
- Inserting and deleting graphic objects such as bitmaps or JPEGs.
- Not destroying objects in VBA by setting the object variables to nothing in the correct sequence so that you avoid "orphaned" objects. In theory this is not necessary, but sometimes problems occur when this is not done. Destroy in inside-out container sequence, for example Range then Worksheet then Workbook.

Excel's UNDO feature does not have a memory leak but will use up memory that could otherwise be used for Formulae.

SAVE removes the UNDO information and frees the memory.

You can reduce the number of Undo levels by editing the Registry. This will tend to reduce the memory used by UNDO: see [Q211922 - XL2000: How to Modify the Number of Undo Levels](#).

On large spreadsheets you should be careful about editing very large numbers of formulae in a single step: edit in small steps with intermediate Saves.

The Excel/VBA Community.

Acknowledgements:

Many people contribute outstandingly and freely to the Excel/VBA community. Particular thanks are owed to:

Rob Bovey: <http://www.appspro.com/>

- VBA Code Cleaner

Stephen Bullen: www.bmsltd.ie

- “Excel 2002 VBA Programmers Reference” with John Green
- VBA Indenter

Ken Getz: KenG@mcwtech.com

- “VBA Developers Handbook” with Mike Gilbert

John Walkenbach: <http://www.j-walk.com/ss/>

- “Excel 2002 Power Programming with VBA”

All the Excel MVP's, with special mentions for:

- Laurent Longre: <http://longre.free.fr/>
- Ture Magnusson: <http://www.turedata.se/>
- David McRitchie:
<http://www.mvps.org/dmccritchie/excel/excel.htm>
- Chip Pearson: <http://www.cpearson.com/>
- Robert Umlas: <mailto:rumlas@kpmg.com>

The participants in [Excel-L](#), the Microsoft Excel Developers List.

Glossary of Terms

Add-In

An externally developed program integrated into Excel.

approximate match

A fast lookup technique used on sorted data that returns the closest match to the lookup-value.

Area

A rectangular block of Formulae.

Array Formula

A special calculation method for an Excel Formula that involves repeated calculation of the formulae for one or more ranges of cells.

Automatic

A calculation option that tells Excel to recalculate the workbook whenever any change is made.

Binary Search

A fast search method for sorted data which works by progressively halving the area of data to be searched.

Bottleneck

Something in your spreadsheet that causes Excel to calculate slowly.

Circular References

A circular chain of references: cell A refers to Cell B which refers to Cell A.

Dependency

When a formula refers to another cell the formula is Dependent on that cell, and must be calculated after that cell.

Drill Down

A method of progressively refining the search for Calculation Bottlenecks

Dynamic Range Names

Excel Defined Names that dynamically resize as the data changes.

exact match

A lookup option in which the lookup gives an error unless an exact match for the lookup-value is found. The data does not need to be sorted.

Excel MVP

The Microsoft Excel MVP program recognises non-Microsoft individuals who have made a consistently excellent and significant contribution to the Excel Newsgroups and Forums.

External Links

Formulae in a workbook which refer to cells in another Workbook.

FastExcel

FastExcel is a product to help you make Excel calculate faster.

You can purchase FastExcel from the Decision Models Website, www.DecisionModels.com

Formats

Each Format defines a unique combination of all the formatting properties of a cell: number format, text format, colour, size etc.

Forward Reference

A reference in a formula to a cell further down this worksheet, or to a cell on a subsequent worksheet.

Forward Worksheet Cross-references

A reference from one worksheet to a sheet which will be calculated later in the Worksheet calculation sequence.

Full Calculation

An Excel calculation method that calculates all the Formulae in all the open workbooks.

Lookup

Searching a table using one or more lookup-values to find the corresponding result value.

Manual

A calculation option that tells Excel not to automatically recalculate the workbook: the user has to request recalculation by pressing F9.

Memory Leaks

Memory used by a program, but not freed after use. The result is a reduction in available memory.

MHZ:

A crude measure of the speed of a PC.

MSKB

Microsoft Knowledge Base: a web-based searchable database containing articles on problems and advice on Microsoft products.

Pivot Cache

A block of memory used to store the information needed by a Pivot Table.

Range Calculate

An Excel calculation method that calculates one or more Areas of Formulae. The calculation method ignores dependencies and calculates each area left-to-right and top-to-bottom

Recalculate

Excel's Smart Calculation method, which only calculates cells which have changed, or which are dependent on other cells whose result value has changed.

Response Time

Response Time is defined as the length of time between a user action, for example pressing a key, and the Computer's response.

Semiautomatic

The same as Automatic, except for Excel Tables.

Sheet Calculate

An Excel calculation Method that calculates only the selected worksheets, and assumes that all other worksheets have been fully calculated.

Sheet Overhead

The difference between the time to calculate the Sheet and the time to calculate all the ranges on the sheet: an indicator of complex sheet structure.

Swap-File

The file on your hard disk that Windows uses for all the Virtual Memory Pages that arent currently needed and have been swapped out of RAM.

Used Range

The rectangular block of cells that includes the last row and column that Excel considers used. Excel's definition of used includes formatting.

User Defined Functions

Worksheet functions that are not built-in to Excel, but have been developed by an Excel user.

Volatile Functions

Functions which are calculated during every calculation.

Wildcard

A way of searching a text string for combinations of characters. In Excel the Wildcard characters ? (any single character) and * (any character or characters) can be used in a number of Functions and operations.

Workbook Overhead

The difference between the time to calculate the workbook and the time to calculate all the worksheets. Can often be reduced by Optimising the Worksheet Calculation Sequence.

Workbook Volatility

The ratio of Recalculation time to Full Calculation time. A high value is caused either by Volatile Functions or more than 65536 references to dependent cells.

Worksheet Calculation Sequence

The sequence in which Excel calculates the worksheets. In Excel97 and Excel2000 this is controlled by the alphabetic sequence of the names of the worksheets. This sequence can have a significant effect on calculation time.

Worksheet Cross-reference

A formula on one worksheet that refers to a cell on a different worksheet.

Index

A

Application.Volatile 35, 53, 57
approximate match 12, 17, 18
Array Formulae 12, 15, 27, 29
Automatic 1, 4, 9, 33–34, 41–42, 44–46, 49–51, 53–54, 60–61, 75
Automatic Calculation 41, 45
Automatic except Tables 4, 41

B

Backward 5
Backward Reference 36
Binary Search 12, 17
Bottlenecks 2, 6, 8, 10–11, 29
Build Time 23
Build Vertically 3

C

Calculation Option 4, 44
Calculation Options 4, 42, 50
Calculation Sequence 2, 11, 32, 35–38, 40, 46, 50, 57, 60, 67–68
character strings 13
Charts 15, 75–76
Circular 32, 37, 44, 67–68
circular references 37, 43–44, 50, 67–68
Code Cleaner 14, 79
Conditional Formats 13, 46
constants 4
contiguous columns 19
Contiguous Rows 19
COUNT 12
COUNTIF 12, 19, 24, 31
Cross-references 32
Ctrl-Alt-F9 33, 41, 45, 56

D

Data on the Same Worksheet 3
Data on a Different Worksheet 3
Data Validation 13
Defined Names 13, 33, 54, 77

Dependencies 33, 35, 37, **Error! Not a valid bookmark in entry on page 39, 56**

Dependency Tree 44

Dependency Tree Limits 44

E

Exact Match 12, 19
External Links 15, 75

F

F9 33–35, 39–41, 44, 45, 47–49, 51, 56–57, 59, 61, 68
FastExcel 2, 5–8, 11–14, 16, 18–20, 22, 24, 27, 29, 32, 38, 40–42, 46, 50–51, 53, 71
Final Results 24
FindFast 7–8
Formats 13, 15
Forward 5, 11, 32, 36, 40
Forward Reference 5, 36
Full Calculation 33, 39, 45, 48, 56
Function 12, 29, 34–35, 37, 39, 53, 55–57, 59–66, 78

G

Goal Seek 67
Graphics 15, 75

H

HLOOKUP 12, 16, 21
Horizontal Worksheet 3

I

INFO 53, 71, 73
Inter Sheet Dependencies 40
Inter Workbook Dependencies 40
Intermediate Results 24
Iteration 37, 41, 43–44, 46, 50, 67–68

L

Limits 44, 69, 72, 76
Lookup 12, 16, 18–21, 31, 44
Lotus 15

M

Manual Calculation 1, 5, 9, 41, 54, 61, 68
Match 12, 18–19
maximum 37, 39, 43, 73–74, 76
Memory 3, 5, 14–15, 29, 69–70, 71–76, 78
Memory Leaks 72–73, 78
memory used 14, 71, 73–74, 77–78
MHZ 3, 7, 30
Microsoft Knowledge Base 3, 72, 76
Missing Values 18–19
Multiple Columns 19, 28
Multiple Worksheets 45

N

natural language 43, 50

O

OFFSET 17, 20, 22, 31, 36–37, 44, 53
Optimise 2, 5, 11, 16, 22, 32, 38, 40, 68
Optimise Worksheet Sequence 32
Option 4, 12, 17, 34, 43–44, 54
Out of Memory 15, 72, 75
Outlook 7–8

P

Phantom Links 11
Pivot 24, 71, 74, 76–77
Precision as Displayed 42, 43
Profile Workbook 12, 14, 71

R

RAM 3, 6, 15, 69–73
Range Calculate 36, 46, 49
Range Calculation 45
Recalculate before Save 43
recalculation 12, 17, 33–35, 37, 39, 41, 42, 44, 45–46,
50, 51, 53–56, 58, 68
reduce 15, 29–31, 44, 70, 77–78
Repeated Calculations 26
Risk Analysis 67
Run-Time 23

S

Scenarios 67
separate sheet 3
Shared Workbook 14
Sheet Calculation 45
Shift-F9 45, 59
size 6, 10, 14, 23, 26, 28, 29, 44, 69, 70–72, 74, 78
Slow Data Entry 5
Smart Recalculation 33
Solver 67
sorted data 12, 17, 18, 25, 29
Subset Addressing 24
SUBTOTAL 12, 25, 54
SUM 12, 24–25, 27–31, 49
SUMIF 12, 24, 27, 30–31
SUMPRODUCT 27–28, 30
Swap-File 6–7, 69, 72

T

Tables 4, 21, 23–25, 41, 67, 74, 77
Totals 24–25

U

UNDO 34, 47, 74, 78
Used Range 8, 14, 53

User-Defined Functions 12, 35, 55

V

Vertical Worksheet 3
Visual Basic 14, 60, 76
VLOOKUP 12, 16–19, 21, 77
Volatile 12, 17, 33, 35, 37, 39, 53, 57
Volatile Actions 34, 53

W

Waste 14
wildcard 21, 24
Wildcard Lookup 21
Workbook Links 11, 32
Workbook Volatility 53

Z

Zoom 15, 75