# simple-talk

DATA, DEV, ADMIN, CHOP SUEY

LATEST SQL, .NET AND SYS ADMIN ARTICLES, OPINION AND NEWS

# Some Original Expressions

Published Friday, May 27, 2011 1:20 AM

## Guest Editorial for Simple-Talk newsletter

*In a guest editorial for the Simple-Talk Newsletter, Phil Factor wonders if we are still likely to find some more novel and unexpected ways of using the newer features of Transact SQL: or maybe in some features that have always been there!*

There can be a great deal of fun to be had in trying out recent features of SQL Expressions to see if they provide new functionality. It is surprisingly rare to find things that couldnâ€™t be done before, but in a different and more cumbersome way; but it is great to experiment or to read of someone else making that discovery. One such recent feature is the â€˜table value constructorâ€™, or â€˜VALUES constructorâ€™, that managed to get into SQL Server 2008 from Standard SQL. This allows you to create derived tables of up to 1000 rows neatly within select statements that consist of lists of row values. E.g.

```
SELECT Old_Welsh, number FROM (VALUES ('Un',1),('Dou',2),('Tri',3),('Petuar',4),('Pimp',5),('Chwech',6),('Seith',7),('Wyth',8),('Nau',9),('Dec',10))
AS WelshWordsToTen (Old_Welsh, number)
```

These values can be expressions that return single values, including, surprisingly, subqueries. You can use this device to create views, or in the USING clause of a MERGE statement. Joe Celko covered this here and here. It can become extraordinarily handy to use once one gets into the way of thinking in these terms, and Iâ€™ve rewritten a lot of routines to use the constructor, but the old way of using UNION can be used the same way, but is a little slower and more long-winded.

The use of scalar SQL subqueries as an expression in a VALUES constructor, and then applied to a MERGE, has got me thinking. It looks very clever, but what use could one put it to? I havenâ€™t seen anything yet that couldnâ€™t be done almost as simply in SQL Server 2000, but Iâ€™m hopeful that someone will come up with a way of solving a tricky problem, just in the same way that a freak of the XML syntax forever made the in-line production of delimited lists from an expression easy, or that a weird XML pirouette could do an elegant pivot-table rotation.

It is in this sort of experimentation where the community of users can make a real contribution. The dissemination of techniques such as the Number, or Tally table, or the unconventional ways that the UPDATE statement can be used, has been rapid due to articles and blogs. However, there is plenty to be done to explore some of the less obvious features of Transact SQL. Even some of the features introduced into SQL Server 2000 are hardly well-known.

Certain operations on data are still awkward to perform in Transact SQL, but we mustnâ€™t, I think, be too ready to state that certain things can only be done in the application layer, or using a CLR routine. With the vast array of features in the product, and with the tools that surround it, I feel that there is generally a way of getting tricky things done. Or should we just stick to our lasts and push anything difficult out into procedural code? Iâ€™d love to know your views.

by Phil Factor

# The 10 Commandments of Good Source Control Management

23 May 2011
by Troy Hunt

> Simple-Talk generally doesn't re-publish anything from another site, but Troy's popular blog post on the Ten Commandments of Source Control was too good to miss. Here is Troy's updated version in the light of the readers' comments made when it was first published.

Ah source control, if there's a more essential tool which indiscriminately spans programming languages without favour, I'm yet to see it. It's an essential component of how so many of us work; the lifeblood of many development teams, if you like. So why do we often get it so wrong? Why are some of the really core, fundamentals of version control systems often so poorly understood?

I boil it down to 10 practices – or "commandments" if you like – which often break down or are not properly understand to begin with. These are all relevant to version control products of all types and programming languages of all flavours. I'll pick some examples from Subversion and .NET but they're broadly applicable to other technologies.

## 1. Stop right now if you're using VSS – *just stop it!*

It's dead. Let it go. No really, it's been on life support for years, taking its dying gasps as younger and fitter VCS tools have rocketed past it. And now it's really seriously about to die as Microsoft finally pulls the plug next year (after several stays of execution).

In all fairness, VSS was a great tool. In 1995. It just simply got eclipsed by tools like Subversion then the distributed guys like Git and Mercurial. Microsoft has clearly signalled its intent to supersede it for many years now – the whole TFS thing wasn't exactly an accident!

The point is that VSS is very broadly, extensively, almost unanimously despised due to a series of major shortcomings by today's standards. Colloquially known as Microsoft's source destruction system, somehow it manages to just keep clinging on to life despite extensively documented glitches, shortcomings and essential functionality (by today's standards), which simply just doesn't exist.

## 2. If it's not in source control, it doesn't exist

Repeat this mantra daily – "The only measure of progress is working code in source control". Until your work makes an appearance in the one true source of code truth – the central source control repository for the project – it simply doesn't exist.

Sure, you've got it secreted away somewhere on your local machine but that's not really doing anyone else any good now, is it? They can't take your version, they can't merge theirs, you can't deploy it (unless you're deploying it wrong) and you're one SSD failure away from losing it all permanently. And just because you're running a distributed version control system doesn't mean you have any protection from that disk failure if it's only been checked into your local branch.

Once you take the mindset of it not existing until it's committed, a whole bunch of other good practices start to fall into place. You break tasks into smaller units so you can commit atomically. You integrate more frequently. You insure yourself against those pesky local hardware failures.

But more importantly (at least for your team lead), you show that you're actually producing something. Declining burn-down charts or ticked-off tasks lists are great, but what do they actually reconcile with? Unless they correlate with working code in source control, they mean nothing.

## 3. Commit early, commit often and don't spare the horses

Further to the previous point, the only way to avoid "ghost code", that which only you can see on your local machine, is to get it into VCS early and often; and don't spare the horses. Addressing the issues from the previous point is one thing that the early and often approach achieves, but here are a few others which can make a significant difference to the way you work:

1. **Every committed revision gives you a rollback position**. If you screw up fundamentally (don't lie, we all do!), are you rolling back one hour of changes or one week?
2. **The risk of a merge nightmare increases dramatically with time.** Merging is never fun. Ever. When you've not committed code for days and you suddenly realise you've got 50 conflicts with other people's changes, you're not going to be a happy camper.
3. **It forces you to isolate features into discrete units of work.** Let's say you've got a 3 man-day feature to build. Oftentimes people won't commit until the end of that period because they're trying to build the whole box and dice into one logical unit. Of course a task as large as this is inevitably comprised of smaller, discrete functions and committing frequently forces you to identify each of these, build them one by one, and commit them to VCS.

When you work this way, your commit history inevitably starts to resemble a semi-regular pattern of multiple commits each work day. Of course it's not always going to be a consistent pattern, there are times we stop and refactor or go through testing phases or any other manner of perfectly legitimate activities which interrupt the normal development cycle.

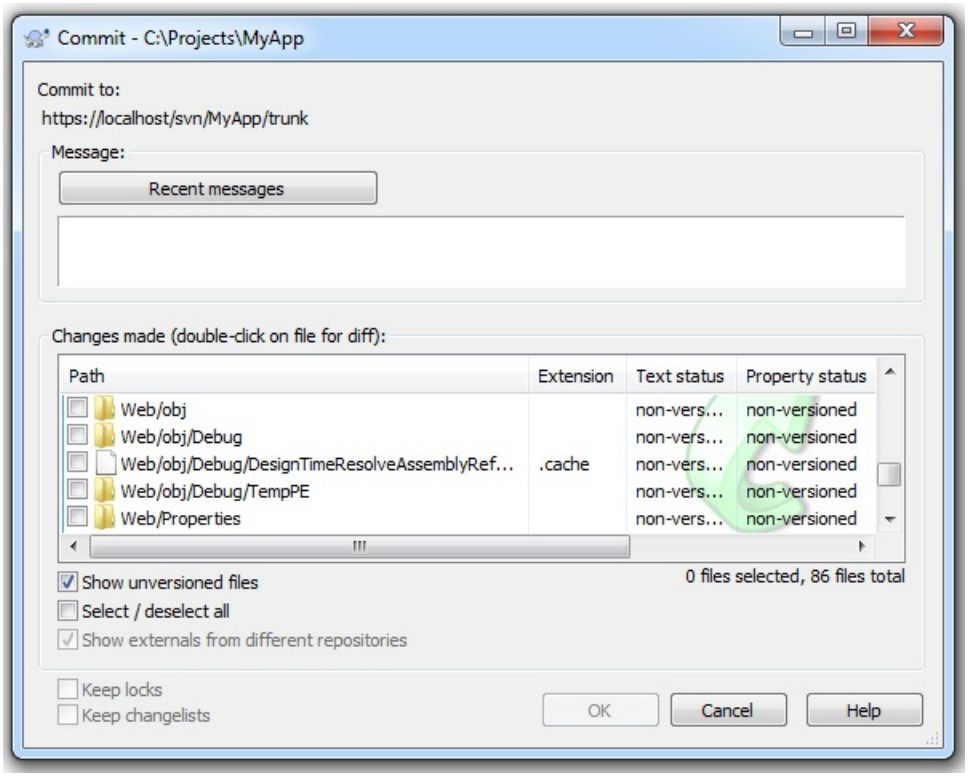
However, when I see an individual – and particularly an entire project – where I know we should be in a normal development cycle and there are

entire days or even multiple days where nothing is happening, I get very worried. I'm worried because as per the previous point, no measurable work has been done but I'm also worried because it usually means something is wrong. Often development is happening in a very "boil the ocean" sort of way (i.e. trying to do everything at once) or absolutely nothing of value is happening at all because people are stuck on a problem. Either way, something is wrong and source control is waving a big red flag to let you know.

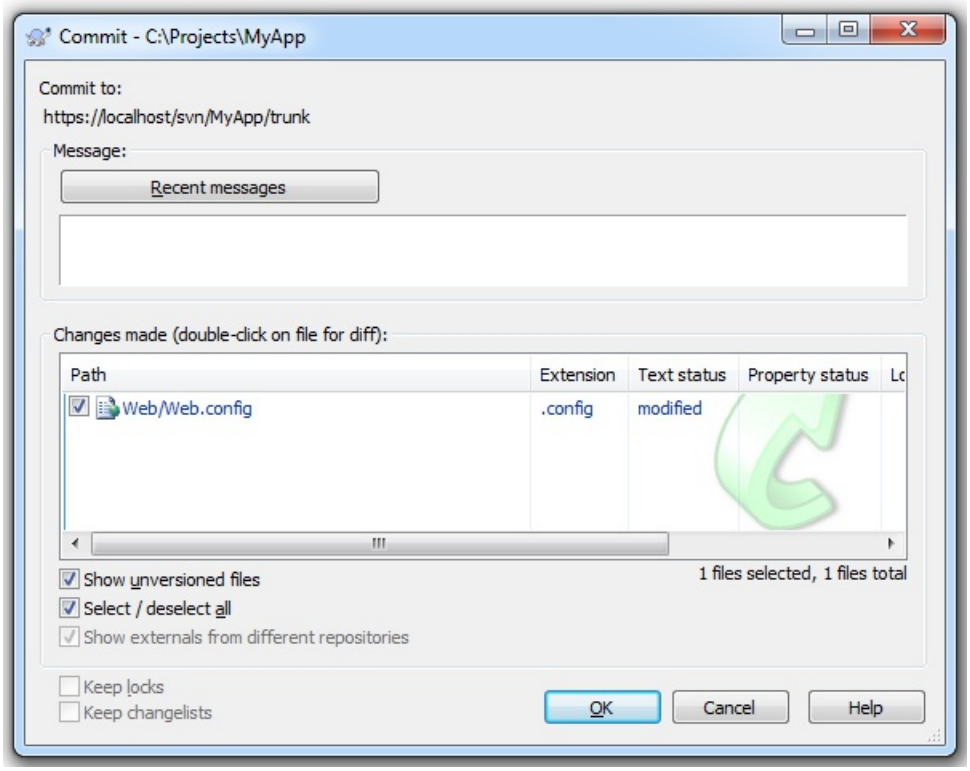
## 4. Always inspect your changes before committing

Committing code into source control is easy; too easy! (Makes you wonder why the previous point seems to be so hard.) Anyway, what you end up with is changes and files being committed with reckless abandon. "There's a change somewhere beneath my project root – quick – get it committed!"

What happens is one (or both) of two things: Firstly, people inadvertently end up with a whole bunch of junk files in the repository. Someone sees a window like the one below, clicks "Select all" and bingo; the repository gets polluted with things like debug folders and other junk that shouldn't be in there.

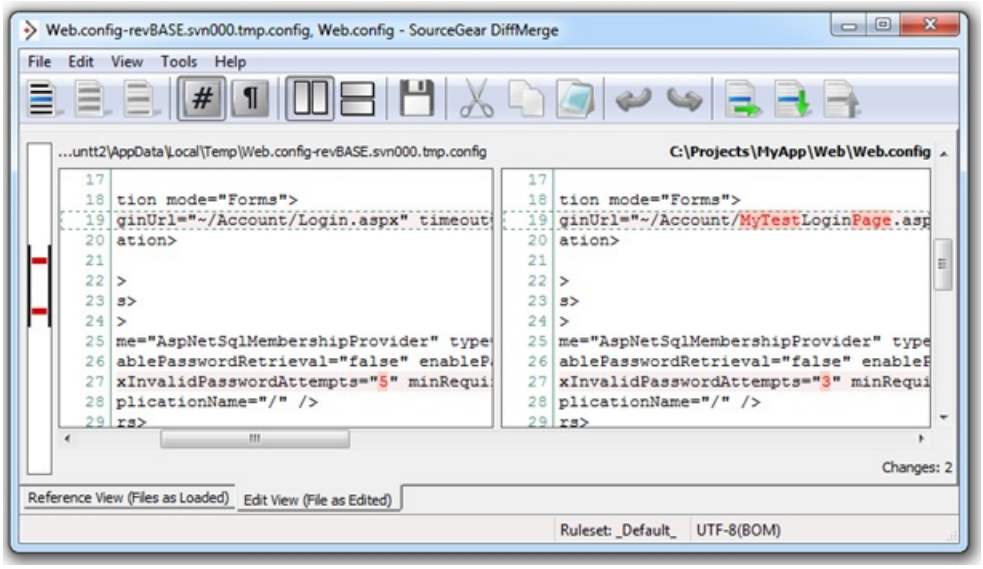



Or secondly, people commit files without checking what they've actually changed. This is real easy to do once you get things like configuration or project definition files where there are a lot going on at once. It makes it really easy to inadvertently put things into the repository that simply weren't intended to be committed and then of course they're quite possibly taken down by other developers. Can you really remember *everything* you changed in that config file?




The solution is simple: **you must inspect each change immediately before committing**. This is easier than it sounds; honest. The whole "inadvertently committed file" thing can be largely mitigated by using the "ignore" feature many systems implement. You never want to commit the Thumbs.db file so just ignore it and be done with it. You also may not want to commit every file that has changed in each revision – so don't!

As for changes within files, you've usually got a pretty nifty diff function in there somewhere. Why am I committing that Web.config file again?



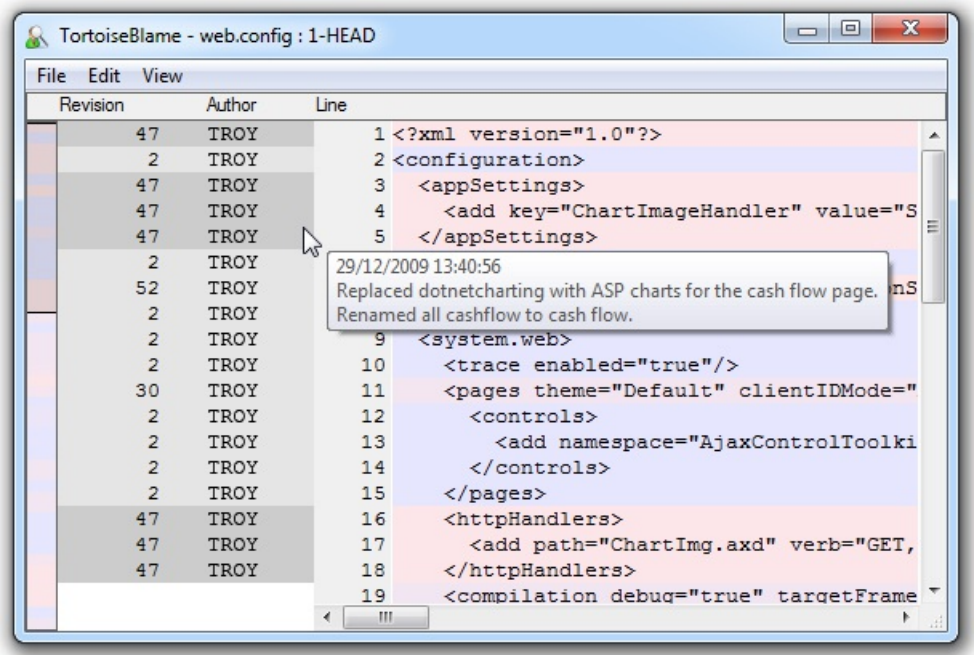

Ah, I remember now, I wanted to decrease the maximum invalid password attempts from 5 down to 3. Oh, and I played around with a dummy login page which I definitely don't want to put into the repository. This practice of pre-commit inspection also makes it much easier when you come to the next section…

## 5. Remember the axe-murderer when writing 'commit' messages

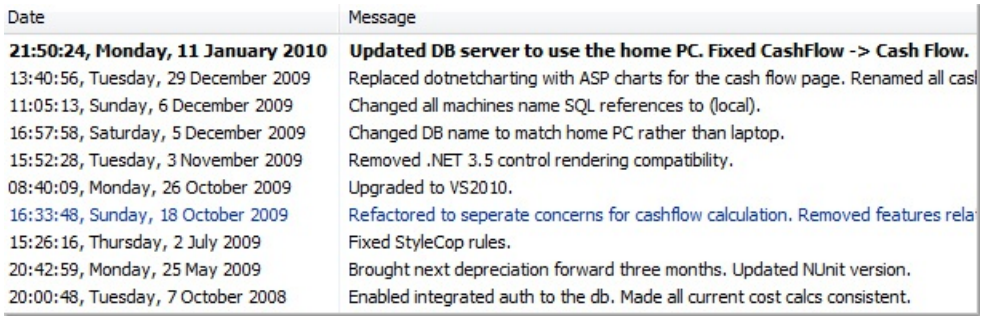
There's an old adage (source unknown), along the lines of "Write every 'commit' message like the next person who reads it is an axe-wielding maniac who knows where you live". If I was that maniac and I'm delving through reams of your code trying to track down a bug and all I can understand from your 'commit' message is "updated some codes", look out, I'm coming after you!

***The whole idea of 'commit' messages is to explain why you committed the code.*** Every time you make any change to code, you're doing it for a reason. Maybe something was broken. Maybe the customer didn't like the colour scheme. Maybe you're just tweaking the build configuration. Whatever it is, there's a reason for it and you need to leave this behind you.

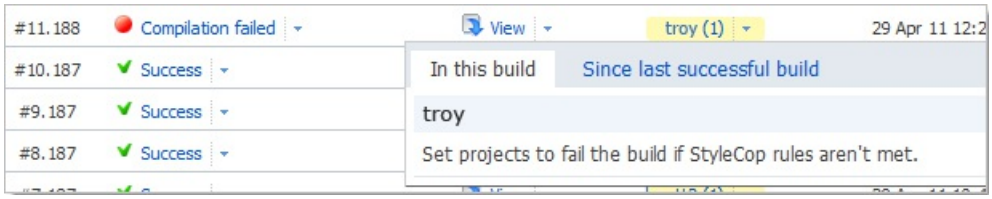
Why? Well there are a few different reasons and they differ depending on the context. For example, using a "blame" feature or other similar functionality which exposes who changed what and hopefully, why. I can't remember what I was doing in the Web.config of this project 18 months ago or why I was mucking around with app settings, but because I left a decent 'commit' message, it all becomes very simple:




It's a similar thing for looking at changes over time. Whether I want to see the entire history of a file, like below, or I just want to see what the team accomplished yesterday, having a descriptive paper trail of comments means it doesn't take much more than a casual glance to get an idea of what's going on.




And finally, 'commit' messages are absolutely invaluable when it comes to tracking down errors. For example, getting to the bottom of why the build is breaking in your continuous integration environment. Obviously my example is overtly obvious, but the point is that bringing this information to the surface can turn tricky problems into absolute no-brainers.




With this in mind, here are some anti-patterns of good commit messages:

1. Some sh*t.
2. It works!
3. fix some flaming errors
4. fix
5. Fixed a little bug...
6. Updated
7. typo
8. Revision 1024!!

Ok, I picked these all out of the Stack Overflow question about What is the WORST commit message you have ever authored, but the thing is that none of them are that dissimilar to many of the messages I've seen in the past. They tell you *absolutely nothing* about what has actually happened in the code; they're junk messages.

One last thing about commit messages; ***subsequent commit messages from the same author should never be identical.*** The reason is simple: you're only committing to source control because something has changed since the previous version. Your code is now in a different state to that previous version and if your commit message is accurate and complete, it logically cannot be identical. Besides, if it was identical (perhaps there's a legitimate edge-case there somewhere), the log is now a bit of a mess to read as there's no way to discern the difference between the two commits.

## 6. You must commit your own changes – you can't delegate it

As weird as this sounds, it happens and I've seen it more than once, most recently just last week. What's happening here is that the source control repository is being placed on a pedestal. For various reasons, the team is viewing it as this sanitised, pristine environment of perfect code. In order to maintain this holy state, code is only committed by a lead developer who carefully aggregates, reviews and (assumedly) tweaks and improves the code before it's committed.

It's pretty easy to observe this pattern from a distance. Very infrequent commits (perhaps weekly), only a single author out of a team with multiple developers and inevitably, conflict chaos if anyone else has gone near the project during that lengthy no-commit period. Very, very nasty stuff.

There are two major things wrong here: Firstly, source control in not meant to be this virginal, unmolested stash of pristine code; at least not throughout development cycles. It's meant to be a place where the team integrates frequently, rolls back when things go wrong and generally comes together around a single common base. It doesn't have to be perfect throughout this process, it only has to (try to) achieve that state at release points in the application lifecycle.

The other problem, and this is the one that really blow me away, is that from the developer's perspective, ***this model means you have no source control!*** It means no integration with code from peers, no rollback, no blame log, no nothing! You're just sitting there in your little silo writing code and waiting to hand it off to the boss at some arbitrary point in the future.

Don't do this. Ever.

## 7. Versioning your database isn't optional

This is one of those ones that everyone knows they should be doing but very often they just file it away in the "too hard" basket. The problem you've got is that many (most?) applications simply won't run without their database. If you're not versioning the database, what you end up with is an incomplete picture of the application which in practice is rendered entirely useless.

Most VCS systems work by simply versioning files on the file system. That's just fine for your typical app files like HTML page, images, CSS, project configuration files and anything else that sits on the file system in nice discrete little units. Problem is that's not quite the way relational databases work. Instead, you end up with these big old data and log files which encompass a whole bunch of different objects and data. This is pretty messy stuff when it comes to version control.

What changes the proposition of database versioning these days is the accessibility of tools like the very excellent SQL Source Control from Red Gate. I wrote about this in detail last year in the post about Rocking your SQL Source Control world with Red Gate so I won't delve into the detail again; suffice to say that ***database versioning is now easy!***


Honestly, if you're not versioning your databases by now you're carrying a lot of risk in your development for no good reason. You have no single source of truth, no rollback position and no easy collaboration with the team when you make changes. Life is just better with the database in source control!

## 8. Compilation output does not belong in source control

Here's an easy way of thinking about it: nothing that is automatically generated as a result of building your project should be in source control. For the .NET folks, this means pretty much everything in the "bin" and "obj" folders which will usually be .dll and .pdb files.

Why? Because if you do this, your co-workers will hate you. It means that every time they pull down a change from VCS they're overwriting their own compiled output with yours. This is both a merge nightmare (you simply can't do it), plus it may break things until they next recompile. And then once they do recompile and recommit, the whole damn problem just gets repeated in the opposite direction and this time you're on the receiving end. Kind of serves you right, but this is not where we want to be.

Of course the other problem is that it's just wasteful. It's wasted on the source control machine disk, it's wasted in bandwidth and additional latency every time you need to send it across the network and it's sure as hell a waste of your time every time you've got to deal with the inevitable conflicts that this practice produces.

So we're back to the "ignore" patterns mentioned earlier on. Once paths such as "bin" and "obj" are set to ignore, everything gets really, really simple. Do it once, commit the rule and everybody is happy.

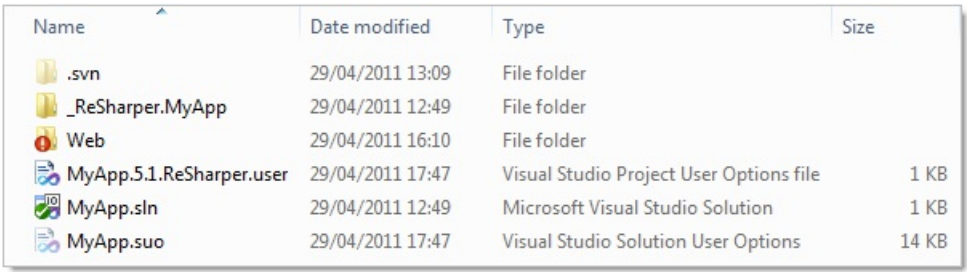
In fact I've even gone so far as to write pre-commit hooks that execute on the VCS server just so this sort of content never makes it into source control to begin with. Sure, it can be a little obtrusive getting your hand slapped by VCS but, well, it only happens when you deserve it! Besides, I'd far rather put the inconvenience back on the perpetrator rather than pass it on to the entire team by causing everyone to have conflicts when they next update.

Some people will argue that their object code belongs in VCS either because they want to retain a permanent history of a particular build or because not everyone can run a particular build so they need it in a commonly accessible location. This, dear people, is what a build server is for. This is the place that compilation history (among other things) is meant to be stored as perpetually accessible artefacts. VCS is not that place for all the reasons mentioned above.

## 9. Nobody else cares about your personal user settings

To be honest, I think that quite often people aren't even aware they're committing their own personal settings into source control. Here's what the problem is: many tools will produce artefacts which manage your own personal, local configurations. They're only intended to be for you and they'll usually be different to everyone else's. If you put them into VCS, suddenly you're all overwriting each other's personal settings. This is not good.

Here's an example of a typical .NET app:



| Name | Date modified | Type | Size |
|---|---|---|---|
| .svn | 29/04/2011 13:09 | File folder | |
| _ReSharper.MyApp | 29/04/2011 12:49 | File folder | |
| Web | 29/04/2011 16:10 | File folder | |
| MyApp.5.1.ReSharper.user | 29/04/2011 17:47 | Visual Studio Project User Options file | 1 KB |
| MyApp.sln | 29/04/2011 12:49 | Microsoft Visual Studio Solution | 1 KB |
| MyApp.suo | 29/04/2011 17:47 | Visual Studio Solution User Options | 14 KB |

The giveaway should be the extensions and type descriptions but in case it's not immediately clear, the .ReSharper.user file and the .suo (Solution User Options) file are both, well, yours. They're nobody else's.

Here's why: Let's take a look inside the ReSharper file:

```
<Configuration>
  <SettingsComponent>
    <string />
    <integer />
    <boolean>
      <setting name="SolutionAnalysisEnabled">True</setting>
    </boolean>
  </SettingsComponent>
  <RecentFiles>
    <RecentFiles>
      <File id="F985644D-6F99-43AB-93F5-C1569A66B0A7/f:Web.config"
        caret="1121" fromTop="26" />
      <File id="F985644D-6F99-43AB-93F5-C1569A66B0A7/f:Site.Master.cs"
        caret="0" fromTop="0" />
```

In this example, the fact that I enabled solution analysis is recorded in the user file. That's fine by me, I like it, other people don't. Normally because they've got an aging, bargain basement PC, but I digress. The point is that this is my setting and I shouldn't be forcing it upon everyone else. It's just the same with the recent files node; just because I recently opened these files doesn't mean it should go into someone else's ReSharper history.

Amusing sidenote: the general incompetence of VSS means ignoring .ReSharper.user files is a bit of a problem.

It's a similar story with the .suo file. Whilst there's not much point looking inside it (no pretty XML here, it's all binary), the file records things like the state of the solution explorer, publishing settings and other things that you don't want to go forcing on other people.

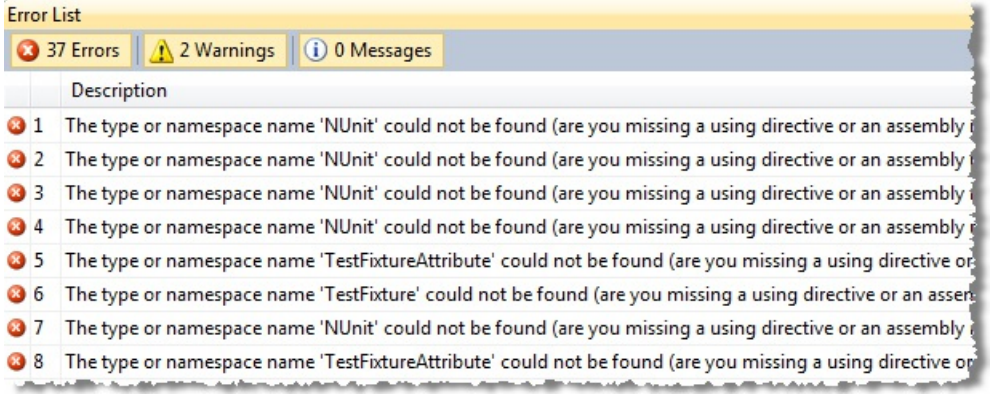
So we're back to simply ignoring these patterns again. At least if you're not running VSS, that is.

## 10. Dependencies need a home too


This might be the last of the Ten Commandments but it's a really, really important one. When an app has external dependencies which are required for it to successfully build and run, ***get them into source control!*** The problem people tend to have is that they get everything behaving real nice in their own little environment with their own settings and their own local dependencies then they commit everything into source control, walk away and think things are cool. And they are, at least until someone else who doesn't have some same local decencies available pulls it down and everything fails catastrophically.

Now just to be crystal clear about the intent of this commandment, it's not so much about getting external assemblies into VCS as it is about ensuring that anyone can pull a project from source control and run it without hunting around for dependencies. If the technology stack you're working with allows you to meet this objective without actually putting assemblies into VCS (i.e. automatically pulls them down from a commonly accessible location), then great!

Moving on, I was reminded of this exact problem myself today when I pulled an old project out of source control and tried to build it:




I'd worked on the assumption that NUnit would always be there on the machine but this time that wasn't the case. Fortunately the very brilliant NuGet bailed me out quickly, but it's not always that easy and it does always take some fiddling when you start discovering that dependencies are missing. In some cases, they're not going to be publicly available and it can be downright painful trying to track them down.

I had this happen just recently where I pulled down a project from source control, went to run it and discovered there was a missing assembly located in a path that began with "c:\Program Files…". I spent literally hours trying to track down the last guy who worked on this (who of course was on the other side of the world), get the assembly, put it in a "Libraries" folder in the project and actually get it into VCS so the next poor sod who comes across the project doesn't go through the same pain.

Of course the other reason this is very important is that if you're working in any sort of continuous integration environment, your build server isn't going to have these libraries installed. Or at least you shouldn't be dependent on it. Doug Rathbone made a good point about this recently when he wrote about Third party tools live in your source control. It's not always possible (and we had some good banter to that effect), but it's usually a pretty easy proposition.

So do everyone a favour and make sure that everything required for your app to actually build and run is in VCS from day 1.

## Summary

None of these things are hard. Honestly, they're really very basic: commit early and often, know what you're committing and that it should actually be in VCS, explain your commits and make sure you do it yourself, don't forget the databases and don't forget the dependencies. But please do forget VSS!

# COM Automation of Office Applications via PowerShell

26 May 2011
by Phil Factor

There need be no shame in using Office by automating it via COM. It was designed to be used that way, and with PowerShell, the various Office applications can be used as glorious output devices for data. Phil Factor uses some practical examples to try to persuade you to take the plunge.

There is something rather satisfying in using Office applications via COM automation in order to cut corners in development work. It can be a very useful way of providing functionality very quickly, and there is a lot of use locked within these applications. In this article, I'll be giving a few illustrations of what can be achieved, with scripts that I find useful. I'll use PowerShell, but you can get the same effect in a similar way in any .NET language, but with rather more effort. I've stripped each script down to the bare minimum so as to make it possible to embed them into this article. We'll end by automatically producing a database build script within a Word document and drawing an Excel graph from the raw data. We'll start a bit simpler, though.

Of course, we'll use automation for silly reasons, such as getting a simple scripted way of generating speech

```
$Excel = New-Object -Com Excel.Application
$Excel.Visible = $false
$Excel.speech.speak('You have failed me, for the last time Admiral.')
$Excel.quit()
```

…or animating the awful animated paper-clip. (In the interests of public safety I won't show you how to do this, but it is great for livening up Powershell presentations. See Jeffrey Snover's code to do this, on page 412 0f Bruce Payette's book 'Powershell in Action'. )

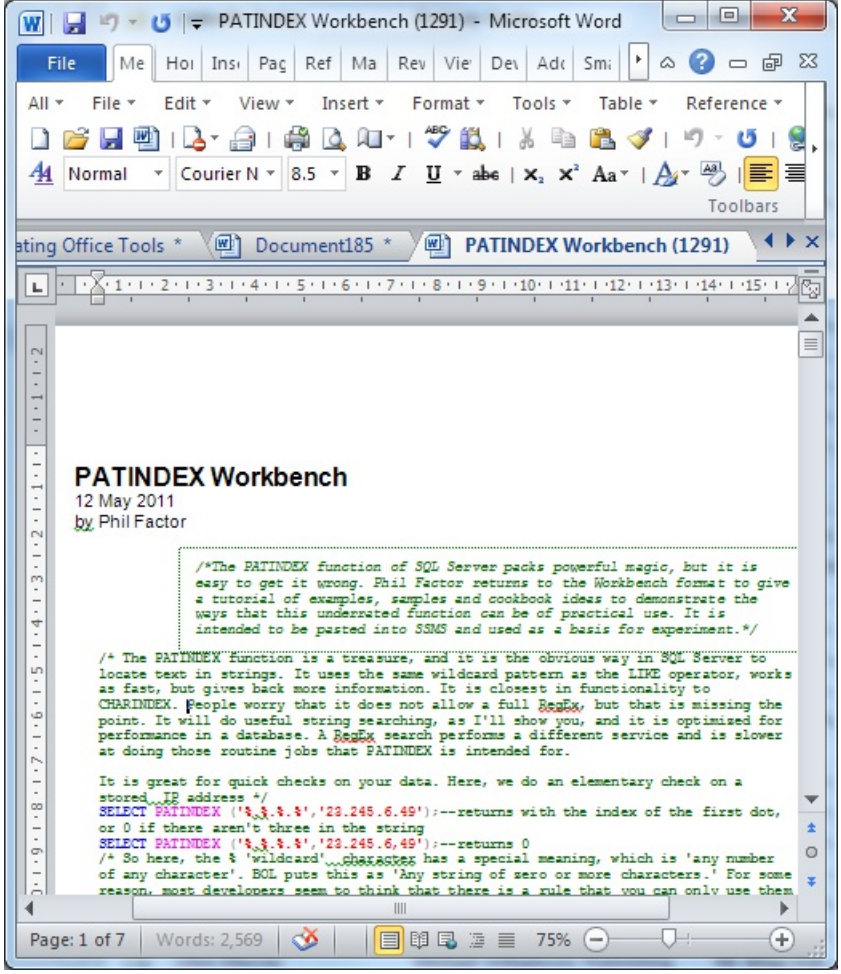
For simple PowerShell scripting of Word and Excel, you'd probably prefer to use **Out-MSWord** by Jeffery Hicks (OUT-MSWord Revised ) for using MS Word as means of output. and the equivalent IMPORT-EXCEL and EXPORT-EXCEL However, we'll try to go beyond this simple sort of usage.

## Scripting MS Word

Microsoft Word is an obvious choice for automation because it is a glorious means of providing output. I once designed a web- application that emailed beautiful PDF invoices using Word. (it read in an HTML file generated by the application). I had it running within hours of an accountant embarrassingly asking 'how does it generate invoices and credit notes' at a presentation. After making it robust and auditable, it remained for the life of the application.

We'll start off with taking a webpage and saving this as a Word document. Actually, for this example we'll save a whole list of articles from Simple-Talk and save them with all their formatting.

```
$DirectoryToSaveTo='s:\work\documents\Test\'
$Word = New-Object -Com Word.Application
$Word.Visible = $false #set this to true for debugging
if (!(Test-Path -path "$DirectoryToSaveTo"))#create it if not existing
  {
  New-Item "$DirectoryToSaveTo" -type directory | out-null
  }
$wdFormatXMLDocument=12 # http://msdn.microsoft.com/en-us/library/bb238158%28v=office.12%29.aspx
# you might want to save it in wdFormatDocument  (value=0) with older versions
foreach ($ArticleID in   @("1233","1291","1289","1290","1288","1287","1286"))
  {
  $Doc = $Word.Documents.Open("http://www.simple-talk.com/content/print.aspx?article=$ArticleID")
  $filename = $DirectoryToSaveTo+"Simple-Talk ("+$ArticleID+').docx' #save it according to its title
  if (test-path $filename) { rm $filename } #delete the file if it already exists
  $Doc.SaveAs([ref]$filename, [ref]$wdFormatXMLDocument)
  $Doc.Close() #close the document
  }
$Word.Quit() #and the instance of Word
```




Of course, you can save in a variety of formats, such as HTML, Text or PDF (if you have the necessary filter) Although this works pretty well, there is a quirk of Word that means that it sometimes makes a better job of the formatting and conversion if the document is pasted from Internet Explorer. OK. we can cope. By modifying the routine, It gives us some advantages because we can get more information about the file at the same time, such as the title of the document.

```
$DirectoryToSaveTo='s:\work\documents\Test\'
$Word = New-Object -Com Word.Application
$Exploder = New-Object -Com InternetExplorer.Application
$Word.Visible = $true
$Exploder.Visible=$true
if (!(Test-Path -path "$DirectoryToSaveTo"))#create it if not existing
  {
  New-Item "$DirectoryToSaveTo" -type directory | out-null
  }
foreach ($ArticleID in   @("1233","1291","1289","1290","1288","1287","1286"))
  {
  $Doc = $Word.Documents.Add() #create a new document
  $Exploder.Navigate("http://www.simple-talk.com/content/print.aspx?article=$ArticleID")
  while ($Exploder.Busy) {} #this should be done with a listener on the onload event
  #OK, so we can now declare some constants.
  $OLECMDID_SELECTALL=17 # see http://msdn.microsoft.com/en-us/library/ms691264%28v=vs.85%29.aspx
  $OLECMDID_COPY=12 # see http://msdn.microsoft.com/en-us/library/ms691264%28v=vs.85%29.aspx
  $wdFormatOriginalFormatting=16 #see http://msdn.microsoft.com/en-us/library/bb237976%28v=office.12%29.aspx
  $Exploder.ExecWB($OLECMDID_SELECTALL,0,$null,[ref]$null) #select all the page
  $Exploder.ExecWB($OLECMDID_COPY,0,$null,[ref]$null) #and copy the selection to the clipboard
        $filename=$Exploder.Document.Title
  $filename=$filename -replace  '[\\/\:\.]',' ' #remove characters that can cause problems
  $Word.Selection.PasteAndFormat($wdFormatOriginalFormatting) # Preserves original formatting of the pasted material.
  $filename = $DirectoryToSaveTo+"$filename ("+$ArticleID+').docx' #save it according to its title
  if (test-path $filename ) { rm $filename } #delete the file if it already exists
  $Doc.SaveAs([ref]$filename)
  $Doc.Close()#close the document
  }
$Word.Quit() #and the instance of Word
```

So what have we done? We've automated two key Microsoft products, Internet Explorer, and Microsoft Word. I've just taken a manual process and scripted it

The script....

- Checks whether  the directory where you want to save the Word files actually exists, and if not it then creates it.
- Fires up internet Explorer
- Fires up MS Word

- navigates to the correct page
- Finds the next article number of the article you want to save as a word file
- Creates a new Word file
- Works out he URL and navigates to the page using IE
- Waits until the page loads
- Select all
- Copy the whole rendered page onto the clipboard
- Pastes the rendered page into Microsoft Word
- Works out the title of the file
- Checks to see if it is already saved in the directory. If so it deletes it
- Saves the file and closes it
- Finds the next article to save
- If nothing else to save, quits

Before you creating a script like this, it pays to be very clear about what the process consists of. If the process is using Word, Excel or Access, you can record a manual process as VBA (Visual Basic for Applications), and convert the VBScript files to Powershell. Other components such as IE take more fishing around in Books-On-Line to use.

Why do this elaborate way when MSWord's own file-conversion isn't too bad? The answer is that you may want to assemble parts of different websites, various images, or a collection of HTML fragments, to create a document. Typically, it would be in making an integrated report from a number of places on an intranet. You can, instead of selecting the whole page, select just a part of it, and by selecting and pasting from various places you can concatenate a document. I use this technique to save tables, scripts and results that are already styled and formatted as XHTML fragments into a document. To do this in Word is tedious even if it makes less complicated word files, but who cares. It is quicker for me to do the formatting I want to data and just paste in each fragment to make up the word document. The recipient of the report never seems to complain. You can read fragments into IE or put the parts of the DOM you want onto the clipboard.

You can create HTML, text or PDF versions of your MSWord files very easily. Here, a word file is being saved as a rather voluminous HTML file. I use this technique to save all my word files in an intranet site so I can look through documents rapidly, and read them on a tablet..

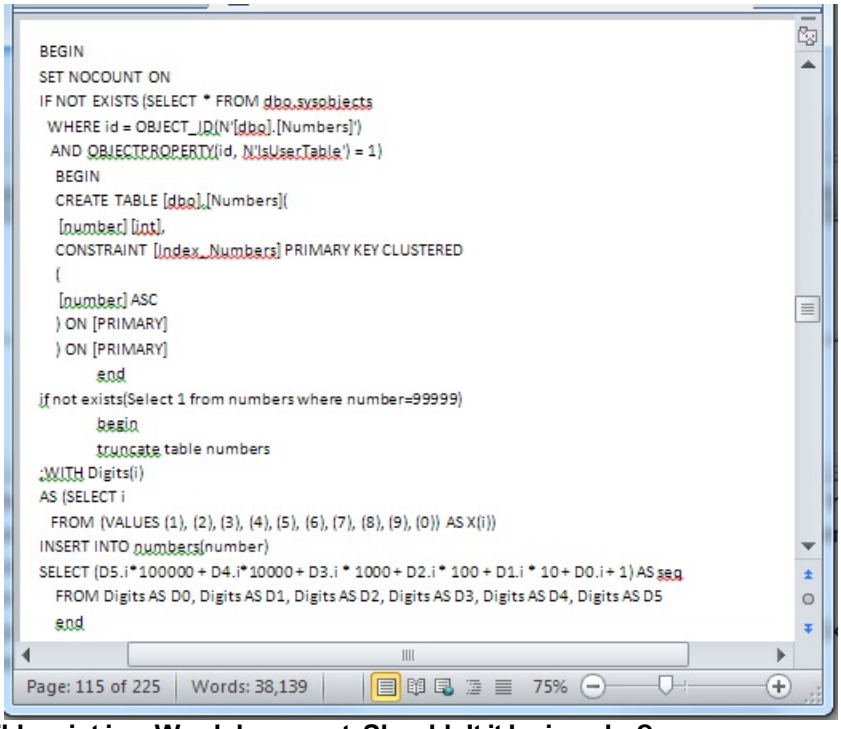
```
$document='MyImportantFile.Docx'
$Word = New-Object -Com Word.Application
$Word.Visible = $false #set to 'true' for debugging!
if (!(test-path $document)) { "No such file as $document"
                              break } # does it exist?
$existingDoc=$word.Documents.Open($document)
$saveaspath = $document.Replace('.docx','.HTML')
<# When you save a Microsoft Word document as HTML, Word displays the Web page similar to the way it will appear in a Web browser.
Formatting and other items that are not supported by HTML or the Web page authoring environment are removed from the file #>
$wdFormatHTML = [ref] 8 #http://msdn.microsoft.com/en-us/library/bb238158%28v=office.12%29.aspx
if (test-path $saveaspath) { rm $saveaspath} #delete the output file if it already exists
$existingDoc.SaveAs( [ref] $saveaspath,$wdFormatHTML )
$existingDoc.Close()
$Word.Quit()
```

To turn to a more practical usage of Word automation, here is a cut-down version of a build-script generator I use to save SQL Server database build scripts as MS Word files. As generating a build script is rather slow, it is best done on the back burner, by running it on the scheduler. I also have a version in SQL Scripts Manager, for doing ad-hoc build scripts.

```
$DirectoryToSaveTo='MyDirectory'
$servername='MyServer'
$database='MyDatabase'
if (!( Test-Path -path "$DirectoryToSaveTo" )) #create it if not existing
  { New-Item "$DirectoryToSaveTo" -type directory | out-null }

$filename="$servername $database" -replace  '[\\\/\:\.]',' ' #remove characters that can cause problems
$filename=$DirectoryToSaveTo+$filename
# Script all the objects in the specified database to a word document
# Load SMO assembly, and  SMOExtended
$v = [System.Reflection.Assembly]::LoadWithPartialName( 'Microsoft.SqlServer.SMO')
if (($v.FullName.Split(','))[1].Split('=')[1].Split('.')[0] -ne '9') {
  [System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SMOExtended') | out-null
}
$s = new-object ('Microsoft.SqlServer.Management.Smo.Server') $servername
$db = $s.databases[$database]
$dbname = $db.Name
$script  = new-object ('Microsoft.SqlServer.Management.Smo.Transfer') ($db)
$script.copyAllObjects = $true #we are going to script everything
$script.Options.ScriptBatchTerminator = $true
$script.copySchema=$true
$sc=$script.scriptTransfer()
"we now have the script. The time has come to write it to Word"
$Word = New-Object -Com Word.Application
$Word.Visible = $true
$Doc = $Word.Documents.Add()
$TitlePara = $Doc.Paragraphs.Add()
$TitlePara.Range.Style = "Heading 1"
$TitlePara.Range.Text = "/* Build Script for $dbname on $servername */"
$TitlePara.Range.InsertParagraphAfter()
```

```
foreach ($paragraph in $sc)  {
  $Para2 = $Doc.Paragraphs.Add()
  $Para2.Range.Text = $paragraph
  $Para2.Range.InsertParagraphAfter()
  }
$Word.Selection.WholeStory()
$Word.Selection.paragraphFormat.SpaceBefore = 0
$Word.Selection.paragraphFormat.SpaceAfter = 0
if (test-path $filename+'.docx' ) { rm $filename+'.docx'} #delete the output file if it already exists
if (test-path $filename+'.doc' ) { rm $filename+'.doc'} #even in the old format.
$Doc.SaveAs([ref]$filename)
$Doc.Close()
$Word.Quit()
"we have completed the task, master."
```




**Just page 115 of 225! The build script in a Word document. Shouldn't it be in color?**

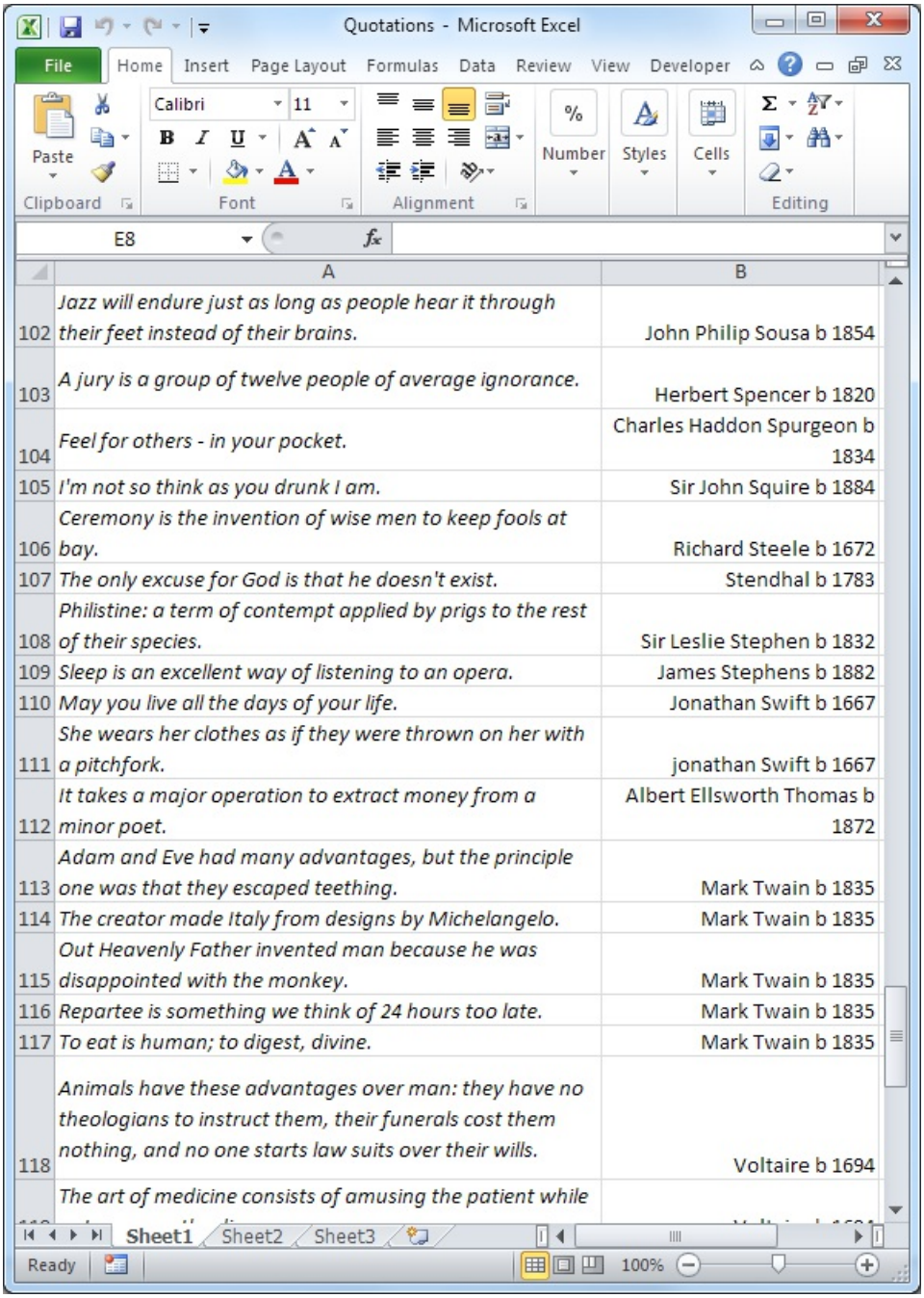
You'll see that the process is extraordinarily simple. As everything is formatted the same way, I do that at the end of the import, simply by highlighting everything and formatting everything at once; automatically, of course.

# Automating Excel

For me, the easiest way of getting data into Excel is by pasting it in via Internet Explorer, or reading it in as an HTML table fragment. It can be done by iterating through the cells via script, but it is a relatively slow way of doing it. If you have a dataSet or dataTable, you can convert it to an HTML table very easily in PowerShell. There are other ways such as writing to the ExcelXML format via the .NET SDK. No, me neither, I'll stick to the wild way. Here is a script that illustrates the 'wild man' approach to pasting data into Excel. it is almost identical to the technique I used with Word.

```
$DirectoryToSaveTo='s:\work\spreadsheets\Test\'
$Filename='Quotations'
$Excel = New-Object -Com Excel.Application
$Exploder = New-Object -Com InternetExplorer.Application
$excel.Visible = $True
$Exploder.Visible=$True
if (!(Test-Path -path "$DirectoryToSaveTo"))#create it if not existing
  {
  New-Item "$DirectoryToSaveTo" -type directory | out-null
  }
$wb = $Excel.Workbooks.Add()
$ws = $wb.Worksheets.Item(1)
$Exploder.Navigate("http://www.simple-talk.com/blogbits/philf/quotations4.html")
while ($Exploder.Busy) {} #this should really be done with a listener on the onload event
#OK, so we can now declare some constants.
$OLECMDID_SELECTALL=17 # see http://msdn.microsoft.com/en-us/library/ms691264%28v=vs.85%29.aspx
$OLECMDID_COPY=12 # see http://msdn.microsoft.com/en-us/library/ms691264%28v=vs.85%29.aspx
$Exploder.ExecWB($OLECMDID_SELECTALL,0,$null,[ref]$null) #select all the page
$Exploder.ExecWB($OLECMDID_COPY,0,$null,[ref]$null) #and copy the selection to the clipboard
$xlbottom=[int]-4107 #http://msdn.microsoft.com/en-us/library/microsoft.office.interop.excel.constants%28v=office.14%29.aspx
$xlRight=[int]-4152 #http://msdn.microsoft.com/en-us/library/microsoft.office.interop.excel.constants%28v=office.14%29.aspx
$xlContext=[int]-5002
$xlOpenXMLWorkbook=[int]51 #http://msdn.microsoft.com/en-us/library/bb241279%28v=office.12%29.aspx
$ws.columns.item("A:A").ColumnWidth = 50
$ws.columns.item("B:B").ColumnWidth = 25
$excel.Range("A1").Select()
$excel.ActiveSheet.Paste()
$ws.columns.item("A:A").Select()
```

```
$excel.Selection.Font.Italic = $True
$wb = $Null #set all variables that point to Excel objects to null
$ws = $Null
$Excel=$Null
# hristo deshev's excel trick Pro Windows Powershell p380
[GC]::Collect()"
```




**Our data (quotes in this case, is imported into Excel and formatted automatically**

You'll have noticed that, just to show that we have been able to get in and format the columns that we've entered into Excel, we've changed the widths of the columns and made on italic, and the other one aligned bottom right. Excel has auto-detected the datatype of the values, and generally seems to get this right. (It sometimes needs a little help with dates)

Having proved that it is easy to get data into Excel automatically, we ought to do something more serious with it. In the next example, we'll read in the weather data for Cambridge (UK) from 1961 to 2010. Then we'll create two new computed columns, the first to hold the date, and the second to hold the rainfall in inches rather than the new-fangled centimeters.

Having done that, we'll produce a graph and add a few features, such as a trend-line (the good news is there is no significant change in the rainfall figures)

The objective of all this is to prove that one can generate graphical reports from a simple data set with just a small amount of automation, but with more versatility than one can get from reporting services.

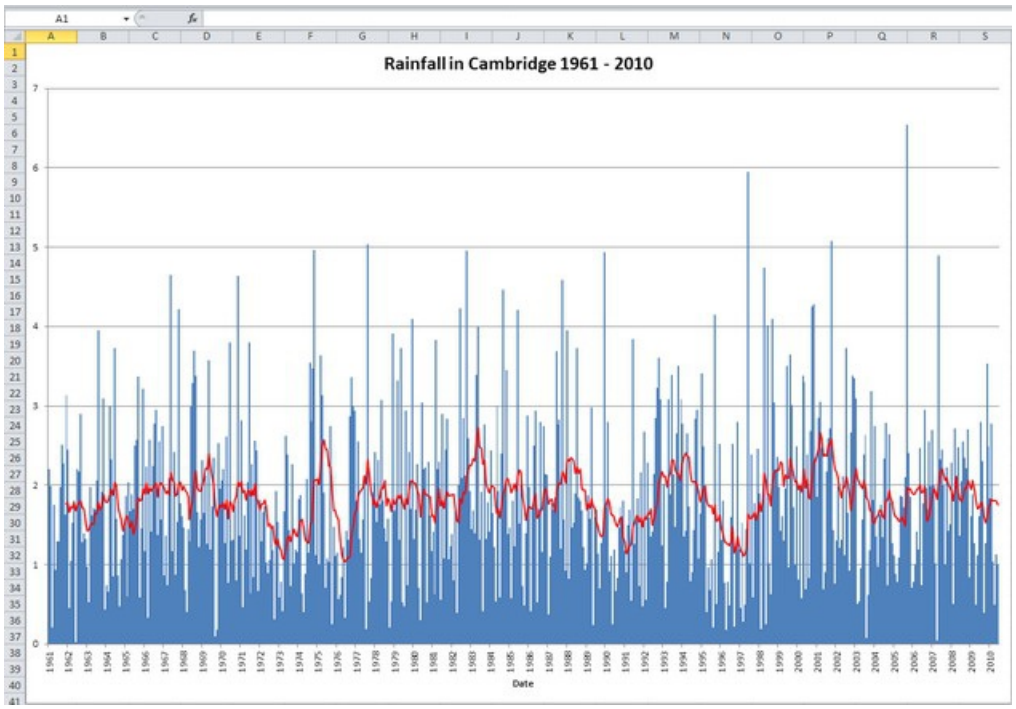
```
$DirectoryToSaveTo='MyDirectory\'
$Filename='CambridgeRainfall'
$Excel = New-Object -Com Excel.Application
$excel.Visible = $True
if (!(Test-Path -path "$DirectoryToSaveTo"))#create it if not existing
```

```
  {
  New-Item "$DirectoryToSaveTo" -type directory | out-null
  }
$wb = $Excel.Workbooks.Open("http://www.simple-talk.com/blogbits/philf/rainfall.html")
$xlbottom=[int]-4107 #http://msdn.microsoft.com/en-us/library/microsoft.office.interop.excel.constants%28v=office.14%29.aspx
$xlRight=[int]-4152 #http://msdn.microsoft.com/en-us/library/microsoft.office.interop.excel.constants%28v=office.14%29.aspx
$xlContext=[int]-5002
$xlBarStacked = 58
$xlCategory = [int]1 #http://msdn.microsoft.com/en-us/library/ff198060.aspx
$xlValue=[int]2 #http://msdn.microsoft.com/en-us/library/ff198060.aspx
$xlPrimary=[int]1 #http://msdn.microsoft.com/en-us/library/ff196160.aspx
$xlColumnClustered   = [int]51
$msoScaleFromTopLeft=0 #from http://msdn.microsoft.com/en-us/library/aa432670
$msoScaleFromBottomRight= 2 #from http://msdn.microsoft.com/en-us/library/aa432670
$msoElementPrimaryCategoryAxisTitleAdjacentToAxis=[int]301 #http://msdn.microsoft.com/en-us/library/ff864118.aspx
$msoElementPrimaryCategoryAxisTitleRotated=[int]309 #http://msdn.microsoft.com/en-us/library/ff864118.aspx
$xlMovingAvg=6 #http://msdn.microsoft.com/en-us/library/ff192956.aspx
$xlOpenXMLWorkbook=[int]51 #http://msdn.microsoft.com/en-us/library/bb241279%28v=office.12%29.aspx
$ws=$Excel.ActiveSheet
#we now create two calculated columns from the date
#the first co calculate the excel date and the other to convert cm to inches
$excel.Range("H2").Select()
$excel.ActiveCell.FormulaR1C1 = 'Date'
$excel.Range("H3").Select()
$excel.ActiveCell.FormulaR1C1 = '=DATE(RC[-7],RC[-6],1)'
$excel.Range("I2").Select()
$excel.ActiveCell.FormulaR1C1 = "Rain (in)"
$excel.Range("I3").Select()
$excel.ActiveCell.FormulaR1C1 = '=RC[-3]*0.0393700787'
$excel.Range("H3:I596").Select()
$ws=$excel.Worksheets.add([System.Reflection.Missing]::Value,$ws)
$ch = $ws.shapes.addChart().chart
$ch.ChartType = $xlColumnClustered
$range=$excel.Range('Rainfall!$H$2:$I$596')
$ch.SetSourceData($range)
$ch.HasLegend = $False
$ch.ChartTitle.Text = "Rainfall in Cambridge 1961 - 2010"
$excel.ActiveChart.ChartArea.Select
$shape=$ws.Shapes.item("Chart 1")
$shape.ScaleWidth(2.4770833333, $False, $msoScaleFromBottomRight)
$shape.ScaleHeight( 2.1614581511,  $False, $msoScaleFromBottomRight)
$shape.ScaleWidth( 1.0218671152,  $False, $msoScaleFromTopLeft)
$shape.ScaleHeight( 1.3054874177,  $False, $msoScaleFromTopLeft)
$ch.Axes($xlCategory).MajorUnit = 12
$ch.Axes($xlCategory).TickLabels.NumberFormat = "yyyy;@"
$ch.ChartGroups(1).Overlap = -100
$ch.ChartGroups(1).GapWidth = 0
$ch.ChartGroups(1).varyByCategories = $False
$ch.SeriesCollection(1).Format.Fill.Transparency = 0.599999994
$trendline=$ch.SeriesCollection(1).Trendlines().Add($xlMovingAvg)
$trendline.Type = $xlMovingAvg
$trendline.Period = 12
$trendline.Format.Line.ForeColor.RGB = 255 #http://technet.microsoft.com/en-us/library/ee176944.aspx
$trendline.Format.Line.Weight = 1.75
$ch.SetElement($msoElementPrimaryCategoryAxisTitleAdjacentToAxis)
$ch.Axes($xlCategory, $xlPrimary).AxisTitle.Text = 'Date'
$filename=$filename -replace  '[\\\/\:\.]',' ' #remove characters that can cause problems
$filename = $DirectoryToSaveTo+$filename+'.xlsx' #save it according to its title
if (test-path $filename ) { rm $filename } #delete the file if it already exists
$wb.SaveAs($filename,  $xlOpenXMLWorkbook)
$wb.Saved = $True
$wb.Close() #close the document
$Excel.Quit() #and the instance of Excel
$wb = $Null #set all variables that point to Excel objects to null
$ws = $Null
$Excel=$Null
# Hristo Deshev's Excel trick 'Pro Windows Powershell' p380
[GC]::Collect()
```

L3

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Weather records in Cambridge 1961- 2010 | | | | | | | | |
| 2 | yyyy | mth | tmax (degC) | tmin (degC) | af (days) | rain (mm) | sun (hours) | Date | Rain (in) |
| 3 | 1961 | 1 | 6.2 | 1 | 5 | 55.9 | 50.5 | 01/01/1961 | 2.200787 |
| 4 | 1961 | 2 | 10.3 | 4.3 | 0 | 50.4 | 65.9 | 01/02/1961 | 1.984252 |
| 5 | 1961 | 3 | 13.7 | 3.3 | 1 | 5.2 | 170.4 | 01/03/1961 | 0.204724 |
| 6 | 1961 | 4 | 14.9 | 6.2 | 0 | 44.5 | 101 | 01/04/1961 | 1.751969 |
| 7 | 1961 | 5 | 16.4 | 6.4 | 0 | 23.6 | 218.4 | 01/05/1961 | 0.929134 |
| 8 | 1961 | 6 | 21.4 | 9.8 | 0 | 32.7 | 243.2 | 01/06/1961 | 1.287402 |
| 9 | 1961 | 7 | 20.9 | 11.3 | 0 | 32.8 | 185.6 | 01/07/1961 | 1.291339 |
| 10 | 1961 | 8 | 21.3 | 11.6 | 0 | 50.3 | 182.6 | 01/08/1961 | 1.980315 |
| 11 | 1961 | 9 | 20.4 | 11.5 | 0 | 63.8 | 131.1 | 01/09/1961 | 2.511811 |
| 12 | 1961 | 10 | 15.3 | 7.3 | 0 | 57.8 | 141.4 | 01/10/1961 | 2.275591 |
| 13 | 1961 | 11 | 9.4 | 2.9 | 6 | 41.5 | 72.7 | 01/11/1961 | 1.633858 |
| 14 | 1961 | 12 | 5.5 | -0.6 | 18 | 79.7 | 58.3 | 01/12/1961 | 3.137795 |
| 15 | 1962 | 1 | 7.3 | 0.7 | 9 | 62.3 | 73.5 | 01/01/1962 | 2.452756 |
| 16 | 1962 | 2 | 7.6 | 1.2 | 9 | 11.4 | 79.3 | 01/02/1962 | 0.448819 |
| 17 | 1962 | 3 | 6.8 | -1.4 | 17 | 26.5 | 102.4 | 01/03/1962 | 1.043307 |
| 18 | 1962 | 4 | 12.3 | 4.1 | 1 | 38.8 | 139.5 | 01/04/1962 | 1.527559 |
| 19 | 1962 | 5 | 14.5 | 6.6 | 1 | 45.9 | 160.1 | 01/05/1962 | 1.807087 |
| 20 | 1962 | 6 | 19.6 | 8 | 0 | 0.5 | 289.6 | 01/06/1962 | 0.019685 |
| 21 | 1962 | 7 | 19.4 | 11 | 0 | 56 | 126.4 | 01/07/1962 | 2.204724 |
| 22 | 1962 | 8 | 19.9 | 11 | 0 | 55.1 | 182.1 | 01/08/1962 | 2.169291 |
| 23 | 1962 | 9 | 17.5 | 8.9 | 0 | 73.6 | 129.6 | 01/09/1962 | 2.897638 |
| 24 | 1962 | 10 | 14.9 | 6.5 | 1 | 32.5 | 104.4 | 01/10/1962 | 1.279528 |
| 25 | 1962 | 11 | 8.2 | 2.6 | 8 | 35.5 | 34.2 | 01/11/1962 | 1.397638 |
| 26 | 1962 | 12 | 4.6 | -2.5 | 18 | 33.7 | 68.6 | 01/12/1962 | 1.326772 |
| 27 | 1963 | 1 | -0.1 | -5.7 | 29 | 24.7 | 58 | 01/01/1963 | 0.972441 |
| 28 | 1963 | 2 | 1.2 | -3.5 | 27 | 13.2 | 59 | 01/02/1963 | 0.519685 |
| 29 | 1963 | 3 | 9.9 | 2.8 | 7 | 50.3 | 102.6 | 01/03/1963 | 1.980315 |
| 30 | 1963 | 4 | 13.4 | 4.7 | 3 | 41 | 124.9 | 01/04/1963 | 1.614173 |
| 31 | 1963 | 5 | 15.8 | 6.3 | 0 | 43.5 | 189.5 | 01/05/1963 | 1.712598 |

**Here is the imported data and the two added columns that calculate the date and the rainfall in inches**




**The automatically-generated Excel graph of rainfall. Click on it to see it full-size.**

# A few traps for the unwary.

Most automation examples are in VBS. Not many have been converted to PowerShell yet. None of the Office applications will record your actions as PowerShell scripts yet, which is odd, considering Microsoft's keen adoption of PowerShell: You are stuck with VBA. There are one or two things that can cause puzzlement.

COM collections are indexed through the **Item** parameterised-property. Visual Basic uses that Item property as the default if you don't specify it.

PowerShell doesn't, because the general .NET interop method isn't perfect, so occasionally you have to add it. PowerShell uses brackets instead of Visual Basic's parentheses for indexing collections. COM collections follow the Microsoft standard of indexing items in a collection from 1 to n whereas .NET and PowerShell collections use the index 0- n-1 for that. Another problem you'll hit is that methods in VBA don't require the empty parentheses, whereas in PowerShell, they do. You'll spot this problem if you start getting Powershell listing the properties of the object rather than doing the method!

More awkward than this is the problem that the recorded scripts have to follow the logic of the way you approach the GUI. Usually, you work by selecting the object you want to work with and then manipulate its attributes via a series of dialog boxes. This means that scripts tend to work on the same approach even when it isn't the quickest and most efficient way of automating it. I find it is better to get something running fairly quickly and then rework the code to walk the object model rather than work on the selected object.

Sometimes, the documentation you find is pretty rudimentary. Microsoft technical authors for Office Automation are notoriously tight-lipped. Fortunately, Powershell is very good at telling you what is available at any point. An easy way to get to grips with a COM object is to get its members. This is a way of doing it. In this example, we are getting the members exposed for COM automation by Internet Explorer.

```
$Exploder = New-Object -Com InternetExplorer.Application
$exploder | Get-Member| ConvertTo-HTML
$exploder.quit()
```

...Which gives the result...

| Name | MemberType | Definition |
|---|---|---|
| ClientToWindow | Method | void ClientToWindow (int, int) |
| ExecWB | Method | void ExecWB (OLECMDID, OLECMDEXECOPT, Variant, Variant) |
| GetProperty | Method | Variant GetProperty (string) |
| GoBack | Method | void GoBack () |
| GoForward | Method | void GoForward () |
| GoHome | Method | void GoHome () |
| GoSearch | Method | void GoSearch () |
| Navigate | Method | void Navigate (string, Variant, Variant, Variant, Variant) |
| Navigate2 | Method | void Navigate2 (Variant, Variant, Variant, Variant, Variant) |
| PutProperty | Method | void PutProperty (string, Variant) |
| QueryStatusWB | Method | OLECMDF QueryStatusWB (OLECMDID) |
| Quit | Method | void Quit () |
| Refresh | Method | void Refresh () |
| Refresh2 | Method | void Refresh2 (Variant) |
| ShowBrowserBar | Method | void ShowBrowserBar (Variant, Variant, Variant) |
| Stop | Method | void Stop () |
| AddressBar | Property | bool AddressBar () {get} {set} |
| Application | Property | IDispatch Application () {get} |
| Busy | Property | bool Busy () {get} |
| Container | Property | IDispatch Container () {get} |
| Document | Property | IDispatch Document () {get} |
| FullName | Property | string FullName () {get} |
| FullScreen | Property | bool FullScreen () {get} {set} |
| Height | Property | int Height () {get} {set} |
| HWND | Property | int HWND () {get} |
| Left | Property | int Left () {get} {set} |
| LocationName | Property | string LocationName () {get} |
| LocationURL | Property | string LocationURL () {get} |
| MenuBar | Property | bool MenuBar () {get} {set} |
| Name | Property | string Name () {get} |
| Offline | Property | bool Offline () {get} {set} |
| Parent | Property | IDispatch Parent () {get} |
| Path | Property | string Path () {get} |
| ReadyState | Property | tagREADYSTATE ReadyState () {get} |
| RegisterAsBrowser | Property | bool RegisterAsBrowser () {get} {set} |
| RegisterAsDropTarget | Property | bool RegisterAsDropTarget () {get} {set} |
| Resizable | Property | bool Resizable () {get} {set} |
| Silent | Property | bool Silent () {get} {set} |
| StatusBar | Property | bool StatusBar () {get} {set} |
| StatusText | Property | string StatusText () {get} {set} |
| TheaterMode | Property | bool TheaterMode () {get} {set} |
| ToolBar | Property | int ToolBar () {get} {set} |
| Top | Property | int Top () {get} {set} |
| TopLevelContainer | Property | bool TopLevelContainer () {get} |
| Type | Property | string Type () {get} |
| Visible | Property | bool Visible () {get} {set} |
| Width | Property | int Width () {get} {set} |

you may get a "Old format or invalid type library" error when automating Excel. the problem is in your Windows regional settings. If the client computer runs the English version of Excel and the locale for the current user is configured for a language other than English, Excel will try to locate

the language pack for the configured language. If the language pack is not found, the error is reported. To fix this problem, it is best to to install the multilingual user interface pack. (See Microsoft Knowledge Base article http://support.microsoft.com/default.aspx?scid=kb;en-us;320369 )

# Further reading.

For COM automation of office applications, the internet is a frustrating source of wisdom and insight. There are, however, some very good Powershell books by people who know PowerShell very well, with chapters on COM automation.

Pro Windows PowerShell by Hristo Deshev
Windows PowerShell in action by Bruce Payette
Windows PowerShell Cookbook by Lee Holmes

Because it is reasonably searchable, the Technet script repository is a good source of ideas, though the quality of script will vary! There are some other archives with very good COM scripts to Office applications in them, but this is the first port of call.

Excel PowerShell scripts
Outlook PowerShell scripts
Microsoft Word PowerShell scripts
Powerpoint PowerShell scripts
Microsoft Access PowerShell scripts
Visio PowerShell scripts

# Anatomy of a .NET Assembly - Signature encodings

Published Friday, May 27, 2011 12:31 PM

If you've just joined this series, I highly recommend you read the previous posts in this series, starting here, or at least these posts, covering the CLR metadata tables.

Before we look at custom attribute encoding, we first need to have a brief look at how signatures are encoded in an assembly in general.

## Signature types

There are several types of signatures in an assembly, all of which share a common base representation, and are all stored as binary blobs in the `#Blob` heap, referenced by an offset from various metadata tables.

The types of signatures are:

- Method definition and method reference signatures.
- Field signatures
- Property signatures
- Method local variables. These are referenced from the `StandAloneSig` table, which is then referenced by method body headers.
- Generic type specifications. These represent a particular instantiation of a generic type.
- Generic method specifications. Similarly, these represent a particular instantiation of a generic method.

### All these signatures share the same underlying mechanism to represent a type

## Representing a type

All metadata signatures are based around the `ELEMENT_TYPE` structure. This assigns a number to each 'built-in' type in the framework; for example, `Uint16` is `0x07`, `String` is `0x0e`, and `Object` is `0x1c`. Byte codes are also used to indicate SzArrays, multi-dimensional arrays, custom types, and generic type and method variables. However, these require some further information.

Firstly, custom types (ie not one of the built-in types). These require you to specify the 4-byte `TypeDefOrRef` coded token after the `CLASS` (`0x12`) or `VALUETYPE` (`0x11`) element type. This 4-byte value is stored in a compressed format before being written out to disk (for more excruciating details, you can refer to the CLI specification).

SzArrays simply have the array item type after the `SZARRAY` byte (`0x1d`). Multidimensional arrays follow the `ARRAY` element type with a series of compressed integers indicating the number of dimensions, and the size and lower bound of each dimension.

Generic variables are simply followed by the index of the generic variable they refer to.

There are other additions as well, for example, a specific byte value indicates a method parameter passed by reference (`BYREF`), and other values indicating custom modifiers.

## Some examples...

To demonstrate, here's a few examples and what the resulting blobs in the `#Blob` heap will look like. Each name in capitals corresponds to a particular byte value in the `ELEMENT_TYPE` or `CALLCONV` structure, and coded tokens to custom types are represented by the type name in curly brackets.

- A simple field:

```
int intField;

FIELD I4
```

- A field of an array of a generic type parameter (assuming `T` is the first generic parameter of the containing type):

```
T[] genArrayField

FIELD SZARRAY VAR 0
```

- An instance method signature (note how the number of parameters does not include the return type):

```
instance string MyMethod(MyType, int&, bool[][]);

HASTHIS DEFAULT 3
```

```
    STRING
    CLASS {MyType}
    BYREF I4
    SZARRAY SZARRAY BOOLEAN
```

- A generic type instantiation:

```
MyGenericType<MyType, MyStruct>

GENERICINST CLASS {MyGenericType} 2
    CLASS {MyType}
    VALUETYPE {MyStruct}
```

- For more complicated examples, in the following C# type declaration:

```
GenericType<T> : GenericBaseType<object[], T, GenericType<T>> { ... }
```

  the Extends field of the `TypeDef` for GenericType will point to a `TypeSpec` with the following blob:

```
GENERICINST CLASS {GenericBaseType} 3
    SZARRAY OBJECT
    VAR 0
    GENERICINST CLASS {GenericType} 1
        VAR 0
```

- And a static generic method signature (generic parameters on types are referenced using `VAR`, generic parameters on methods using `MVAR`):

```
TResult[] GenericMethod<TInput, TResult>(
    TInput,
    System.Converter<TInput, TOutput>);

GENERIC 2 2
    SZARRAY MVAR 1
    MVAR 0
    GENERICINST CLASS {System.Converter} 2
        MVAR 0
        MVAR 1
```

As you can see, complicated signatures are recursively built up out of quite simple building blocks to represent all the possible variations in a .NET assembly.

Now we've looked at the basics of normal method signatures, in my next post I'll look at custom attribute application signatures, and how they are different to normal signatures.

by Simon Cooper
Filed Under: Anatomy of a .NET Assembly

# Subqueries in SQL Server

26 May 2011
by Robert Sheldon

Subqueries and derived tables can add great versatility to SQL statements, cut down complexity, but can occasionally be a curse when their effect on performance is poorly understood. Surely everyone understands the various types of subqueries and how they are used? If you felt a twinge of doubt, here is Rob Sheldon's easy guide to the subject.

Few elements within a Transact-SQL statement are as versatile as the *subquery*. A subquery—also referred to as an *inner query* or *inner select*—is a SELECT statement embedded within a data manipulation language (DML) statement or nested within another subquery. You can use subqueries in SELECT, INSERT, UPDATE, and DELETE statements wherever expressions are allowed. For instance, you can use a subquery as one of the column expressions in a SELECT list or as a table expression in the FROM clause.

A DML statement that includes a subquery is referred to as the *outer query*. The following guidelines provide details about how to implement subqueries in your outer queries or in other subqueries:

- You must enclose a subquery in parenthesis.
- A subquery must include a SELECT clause and a FROM clause.
- A subquery can include optional WHERE, GROUP BY, and HAVING clauses.
- A subquery cannot include COMPUTE or FOR BROWSE clauses.
- You can include an ORDER BY clause only when a TOP clause is included.
- You can nest subqueries up to 32 levels.

There are several ways you can categorize subqueries—by the number of results they returns, whether they're correlated (linked to the outer query), or where they're used within a DML statement. For the purposes of this article, I take the last approach and explain how subqueries can be implemented in the SELECT, FROM, and WHERE clauses of a SELECT statement. Although you can implement subqueries in other clauses and other statement types, the examples I provide should demonstrate the essential principles of how subqueries can be used in any circumstances. (The examples all return data from the AdventureWorks2008 database on a local instance of SQL Server 2008.)

> **NOTE:** Microsoft documentation states that subqueries perform about the same as statements that are semantically equivalent, such as subqueries and joins. However, if existence must be checked (as will be described later in the article), a join often performs better if the subquery must be processed for each row returned by the outer query.

## Adding Subqueries to the SELECT Clause

You can add a subquery to a SELECT clause as a column expression in the SELECT list. The subquery must return a scalar (single) value for each row returned by the outer query. For example, in the following SELECT statement, I use a subquery to define the TotalQuantity column:

```
SELECT
  SalesOrderNumber,
  SubTotal,
  OrderDate,
  (
    SELECT SUM(OrderQty)
    FROM Sales.SalesOrderDetail
    WHERE SalesOrderID = 43659
  ) AS TotalQuantity
FROM
  Sales.SalesOrderHeader
WHERE
  SalesOrderID = 43659;
```

Notice I've inserted the subquery as the fourth column expression in the SELECT list and named the column TotalQuantity. The subquery itself is enclosed in parentheses and made up of a single SELECT statement. The statement retrieves the total number of items sold for sales order 43659. Because there are multiple line items in this order, I used the SUM aggregate function to add the numbers together and return a single value. The following table shows the result set returned by the outer SELECT statement.

| SalesOrderNumber | SubTotal | OrderDate | TotalQuantity |
|---|---|---|---|
| SO43659 | 24643.9362 | 2001-07-01 00:00:00.000 | 26 |

As the results show, the outer SELECT statement returns a single row from the SalesOrderHeader table for order 43659, and the TotalQuantity column itself returns a value of 26. If you were to run the subquery's SELECT statement on its own (without running the outer query), you would also

receive a value of 26. However, by running the SELECT statement as a subquery within the outer SELECT statement, the total number of items sold is now provided as part of the order information.

You can use a subquery anywhere in a SQL Statement where an expression is allowed. For the next example we'll use it as part of a CASE statement. In the following example, I use a CASE expression and subquery to check whether line item sales totals in the SalesOrderDetail table equals the sales subtotal listed in the SalesOrderHeader table:

```
SELECT
  SalesOrderNumber,
  SubTotal,
  OrderDate,
  CASE WHEN
    (
      SELECT SUM(LineTotal)
      FROM Sales.SalesOrderDetail
      WHERE SalesOrderID = 43659
    ) =  SubTotal THEN 'balanced'
    ELSE 'not balanced'
  END AS LineTotals
FROM
  Sales.SalesOrderHeader
WHERE
  SalesOrderID = 43659;
```

I've included the CASE expression as part of the fourth column expression. The CASE expression uses the subquery to total the line item sales in the SalesOrderDetail table for order 43659. Notice that, as in the preceding example, the subquery is enclosed in parentheses and uses the SUM aggregate function to return a single value. I then use an equal (=) operator to compare the subquery's result to the SubTotal column in the SalesOrderHeader table. If the amounts are equal, the CASE expression returns a value of *balanced*. It the values are not equal, CASE returns *not balanced*. The following table shows the results returned by the outer SELECT statement.

| SalesOrderNumber | SubTotal | OrderDate | LineTotals |
|---|---|---|---|
| SO43659 | 24643.9362 | 2001-07-01 00:00:00.000 | not balanced |

As you can see, the line item sales total in the SalesOrderDetail table does not match the subtotal in the SalesOrderHeader table, at least not for sale 43659. However, suppose you want to verify all the sales listed in the two tables to see whether the totals balance. To do so, you must modify both the subquery and the outer query in order to create the condition necessary to support a *correlated subquery*. A correlated subquery, also known as a *repeating subquery*, is one that depends on the outer query for specific values. This is particularly important if your outer query returns multiple rows.

The best way to understand how correlated subqueries work is to look at an example. In the following SELECT statement, I include a CASE expression as one of the column expressions, as you saw in the preceding example:

```
SELECT
  SalesOrderNumber,
  SubTotal,
  OrderDate,
  CASE WHEN
    (
      SELECT SUM(LineTotal)
      FROM Sales.SalesOrderDetail d
      WHERE d.SalesOrderID = h.SalesOrderID
    ) =  h.SubTotal THEN 'balanced'
    ELSE 'not balanced'
  END AS LineTotals
FROM
  Sales.SalesOrderHeader h;
```

As before, the CASE expression includes a subquery that returns the total amount for line item sales. However, notice that the subquery's WHERE clause is different from the previous example. Instead of specifying an order ID, the WHERE clause references the SalesOrderID column from the outer query. I do this by using table aliases to distinguish the two columns—*h* for SalesOrderHeader and *d* for SalesOrderDetail—and then specifying that the column values must be equal for the WHERE condition to evaluate to true. That means that, for each row in the SalesOrderHeader table returned by the outer query, the SalesOrderID value associated with that row is plugged into the subquery and compared with the SalesOrderID value of the SalesOrderDetail table. As a result, the subquery is executed for each row returned by the outer query.

The value returned by the subquery is then compared to the SubTotal column of the SalesOrderHeader table and a value for the LineTotals column

is provided, a process repeated for each row. The following table provides a sample of the data returned by the outer query.

| SalesOrderNumber | SubTotal | OrderDate | LineTotals |
|---|---|---|---|
| SO61168 | 1170.48 | 2003-12-31 00:00:00.000 | balanced |
| SO61169 | 619.46 | 2003-12-31 00:00:00.000 | balanced |
| SO61170 | 607.96 | 2003-12-31 00:00:00.000 | balanced |
| SO61171 | 553.97 | 2003-12-31 00:00:00.000 | balanced |
| SO61172 | 2398.05 | 2003-12-31 00:00:00.000 | balanced |
| SO61173 | 34851.8445 | 2004-01-01 00:00:00.000 | not balanced |
| SO61174 | 8261.4247 | 2004-01-01 00:00:00.000 | not balanced |
| SO61175 | 30966.9005 | 2004-01-01 00:00:00.000 | not balanced |
| SO61176 | 1570.725 | 2004-01-01 00:00:00.000 | not balanced |
| SO61177 | 25599.8392 | 2004-01-01 00:00:00.000 | not balanced |
| SO61178 | 3227.0112 | 2004-01-01 00:00:00.000 | not balanced |
| SO61179 | 47199.0054 | 2004-01-01 00:00:00.000 | not balanced |
| SO61180 | 4208.8078 | 2004-01-01 00:00:00.000 | not balanced |
| SO61181 | 36564.9023 | 2004-01-01 00:00:00.000 | not balanced |
| SO61182 | 63162.5722 | 2004-01-01 00:00:00.000 | not balanced |
| SO61183 | 35.0935 | 2004-01-01 00:00:00.000 | not balanced |
| SO61184 | 113451.8266 | 2004-01-01 00:00:00.000 | not balanced |
| SO61185 | 554.0328 | 2004-01-01 00:00:00.000 | not balanced |
| SO61186 | 39441.4489 | 2004-01-01 00:00:00.000 | not balanced |
| SO61187 | 65.988 | 2004-01-01 00:00:00.000 | balanced |
| SO61188 | 58992.9256 | 2004-01-01 00:00:00.000 | not balanced |

As you can see, some of the totals balance out, and others do not. Again, the important thing to keep in mind with correlated subqueries is that the subquery is executed for each row returned by the outer query. The correlated subquery then uses a value supplied by the outer query to return its results. For more details about correlated subqueries, see the topic "*Correlated Subqueries*" in SQL Server Books Online.

## Adding Subqueries to the FROM Clause

The subquery examples in the previous section each return a single value, which they must do in order to be used in the SELECT clause. However, not all subquery results are limited in this way. A subquery can also be used in the FROM clause to return multiple rows and columns. The results returned by such a subquery are referred to as a *derived table*. A derived table is useful when you want to work with a subset of data from one or more tables without needing to create a view or temporary table. For instance, in the following example, I create a subquery that retrieves product subcategory information from the ProductSubcategory table, but only for those products that include the word "bike" in their name:

```
SELECT
  p.ProductID,
  p.Name AS ProductName,
  p.ProductSubcategoryID AS SubcategoryID,
  ps.Name AS SubcategoryName
FROM
  Production.Product p INNER JOIN
  (
    SELECT ProductSubcategoryID, Name
    FROM Production.ProductSubcategory
    WHERE Name LIKE '%bikes%'
  ) AS ps
  ON p.ProductSubcategoryID = ps.ProductSubcategoryID;
```

The first thing to notice is that the subquery returns a derived table that includes two columns and multiple rows. Because the subquery returns a table, I can join that table, which I've named *ps*, to the results from the Product table *(p*). As the join demonstrates, you treat a subquery used in the FROM clause just as you would treat any table. I could have just as easily created a view or temporary table—or even added a regular table to the database—that accesses the same data as that available through the subquery.

I defined the join based on the subcategory ID in the derived table and Product table. I was then able to include columns from both these tables in the SELECT list, as I would any type of join. The following table shows a subset of the results returned by the outer query.

| ProductID | PeoductName | SubcategoryID | SubcategoryName |
|---|---|---|---|
| 786 | Mountain-300 Black, 40 | 1 | Mountain Bikes |
| 787 | Mountain-300 Black, 44 | 1 | Mountain Bikes |
| 788 | Mountain-300 Black, 48 | 1 | Mountain Bikes |
| 789 | Road-250 Red, 44 | 2 | Road Bikes |
| 790 | Road-250 Red, 48 | 2 | Road Bikes |
| 791 | Road-250 Red, 52 | 2 | Road Bikes |
| 792 | Road-250 Red, 58 | 2 | Road Bikes |
| 793 | Road-250 Black, 44 | 2 | Road Bikes |
| 794 | Road-250 Black, 48 | 2 | Road Bikes |

| | | | |
|---|---|---|---|
| 795 | Road-250 Black, 52 | 2 | Road Bikes |
| 796 | Road-250 Black, 58 | 2 | Road Bikes |
| 797 | Road-550-W Yellow, 38 | 2 | Road Bikes |
| 798 | Road-550-W Yellow, 40 | 2 | Road Bikes |
| 799 | Road-550-W Yellow, 42 | 2 | Road Bikes |
| 800 | Road-550-W Yellow, 44 | 2 | Road Bikes |
| 801 | Road-550-W Yellow, 48 | 2 | Road Bikes |
| 953 | Touring-2000 Blue, 60 | 3 | Touring Bikes |
| 954 | Touring-1000 Yellow, 46 | 3 | Touring Bikes |
| 955 | Touring-1000 Yellow, 50 | 3 | Touring Bikes |

As you can see, the results include the subcategory names, which are taken from the derived table returned by the subquery. Because I was able to join the Product table to the derived table, I was able to match the subcategory names to the product names in the outer query's result set.

## Adding Subqueries to the WHERE Clause

Another common way of implementing subqueries in a DML statement is to use them to help define conditions in the WHERE clause. For instance, you can use comparison operators to compare a column's value to a value returned by the subquery. In the following example, I use the equal (=) operator to compare the BusinessEntityID value in the Person table to the value returned by a subquery:

```
SELECT
  BusinessEntityID,
  FirstName,
  LastName
FROM
  Person.Person
WHERE
  BusinessEntityID =
  (
    SELECT BusinessEntityID
    FROM HumanResources.Employee
    WHERE NationalIDNumber = '895209680'
  );
```

The subquery retrieves the BusinessEntityID value from the Employee table for the employee whose national ID is 895209680. The BusinessEntityID value from the subquery is then compared to the BusinessEntityID value in the Person table. If the two values are equal, the row is returned, as shown in the following results.

```
SELECT
  p.BusinessEntityID,
  p.FirstName,
  p.LastName,
  s.SalesQuota
FROM
  Person.Person p INNER JOIN
  Sales.SalesPerson s
  ON p.BusinessEntityID = s.BusinessEntityID
WHERE
  s.SalesQuota IS NOT NULL AND
  s.SalesQuota >
  (
    SELECT AVG(SalesQuota)
    FROM Sales.SalesPerson
  );
```

In the subquery, I use the AVG aggregate function to find the average sales quota figure. This way, the subquery returns only one value. I can then compare that value to the SalesQuota column. If the SalesQuota figure is greater than the average, the WHERE expression evaluates to true, and the row is returned by the outer query. Otherwise, the expression evaluates to false and the row is not returned. As the following table shows, only three rows have a SalesQuota value greater than the average.

| BusinessEntityID | FirstName | LastName | SalesQuota |
|---|---|---|---|
| 275 | Michael | Blythe | 300000.00 |
| 279 | Tsvi | Reiter | 300000.00 |
| 284 | Tete | Mensa-Annan | 300000.00 |

At times, you might want to compare your column to a list of values, rather than a single value, in which case you can use one of the following keywords to modify the comparison modifier:

- **ALL:** The column value is compared to all values returned by the subquery.
- **ANY:** The column value is compared to the one most applicable distinct value.
- **SOME:** The ISO equivalent to ANY.

The best way to understand how these modifiers work is to see them in action. In the following example, I use the ANY modifier along with the greater than (>) operator to compare the SalesQuota column to the list of SalesQuota values returned by the subquery:

```
SELECT
  p.BusinessEntityID,
  p.FirstName,
  p.LastName,
  s.SalesQuota
FROM
  Person.Person p INNER JOIN
  Sales.SalesPerson s
  ON p.BusinessEntityID = s.BusinessEntityID
WHERE
  s.SalesQuota IS NOT NULL AND
  s.SalesQuota > ANY
  (
    SELECT SalesQuota
    FROM Sales.SalesPerson
  );
```

In this case, the subquery returns a list of values, rather than one value. I can return a list because I'm using the ANY modifier. As a result, the SalesQuota value for each row returned must be greater than any of the values returned by the subquery. In other words, as long as the SalesQuota value exceeds any one value returned by the subquery, that row is returned. As the following results indicate, only three rows in the SalesPerson table have SalesQuota values that exceed at least one of the values returned by the subquery.

| BusinessEntityID | FirstName | LastName | SalesQuota |
|---|---|---|---|
| 275 | Michael | Blythe | 300000.00 |
| 279 | Tsvi | Reiter | 300000.00 |
| 284 | Tete | Mensa-Annan | 300000.00 |

The next example is identical to the preceding one, except that I use the ALL modifier to qualify the comparison operator:

```
SELECT
  p.BusinessEntityID,
  p.FirstName,
  p.LastName,
  s.SalesQuota
FROM
  Person.Person p INNER JOIN
  Sales.SalesPerson s
  ON p.BusinessEntityID = s.BusinessEntityID
WHERE
  s.SalesQuota IS NOT NULL AND
  s.SalesQuota > ALL
  (
    SELECT SalesQuota
    FROM Sales.SalesPerson
  );
```

Because I've used the ALL modifier, each row returned must have a SalesQuota value that exceeds *all* the values returned by the subquery. In other words, the SalesQuota value must exceed the highest value returned by the subquery. As it turns out, no row has a SalesQuota value that exceeds all the values returned by the subquery, so the statement now returns no rows.

Another operator that lets you work with a subquery that returns a list is the IN operator. The column value is compared to the list, and the WHERE expression evaluates to true if any of the subquery values matches the column value. For example, the following SELECT statement includes a subquery that returns a list of IDs for sales representatives:

```
SELECT
  BusinessEntityID,
  FirstName,
  LastName
FROM
  Person.Person
WHERE
  BusinessEntityID IN
  (
    SELECT BusinessEntityID
    FROM HumanResources.Employee
    WHERE JobTitle = 'Sales Representative'
  );
```

The BusinessEntityID value from the outer query is compared to the list of ID values returned by the subquery. If the BusinessEntityID value matches one of the values in the subquery list, the row is included in the outer query's results, as shown in the following results:

| BusinessEntityID | FirstName | LastName |
|---|---|---|
| 275 | Michael | Blythe |
| 276 | Linda | Mitchell |
| 277 | Jillian | Carson |
| 278 | Garrett | Vargas |
| 279 | Tsvi | Reiter |
| 280 | Pamela | Ansman-Wolfe |
| 281 | Shu | Ito |
| 282 | José | Saraiva |
| 283 | David | Campbell |
| 284 | Tete | Mensa-Annan |
| 286 | Lynn | Tsoflias |
| 288 | Rachel | Valdez |
| 289 | Jae | Pak |
| 290 | Ranjit | Varkey Chudukatil |

If you want to return only those rows whose BusinessEntityID value does not match any values in the list returned by the subquery, you can instead use the NOT IN operator, as in the following example:

```
SELECT
  BusinessEntityID,
  FirstName,
  LastName
FROM
  Person.Person
WHERE
  BusinessEntityID NOT IN
  (
    SELECT BusinessEntityID
    FROM HumanResources.Employee
    WHERE JobTitle = 'Sales Representative'
  );
```

This statement is exactly the same as the preceding example except for the use of the NOT IN operator, but the results are quite different. Rather than returning 14 rows, one for each sales representative, the statement now returns nearly 20,000 rows, one for each person who is not a sales representative.

One other method you can use when including a subquery in your WHERE clause is to check for existence. In this case, you use the EXIST keyword to verify whether the subquery returns a row that matches your search criteria. The subquery doesn't produce any data but instead returns a value of true or false, depending on whether the row exists. For example, in the following SELECT statement, I use a correlated subquery to check the name of each product's subcategory to determine whether that name is Mountain Bikes:

```
SELECT ProductID, Name AS ProductName FROM Production.Product p WHERE EXISTS ( SELECT * FROM
Production.ProductSubcategory s WHERE p.ProductSubcategoryID = s.ProductSubcategoryID AND s.Name =
'Mountain Bikes' );
```

For each row returned by the outer query, the existence of a row returned by the correlated subquery is checked. If a row is returned by the subquery, the existence test evaluates to true, and the outer query's row is included in the result set. The following table shows a partial list of the

results returned by the outer query, after checking for existence.

| ProductID | ProductName |
|---|---|
| 771 | Mountain-100 Silver, 38 |
| 772 | Mountain-100 Silver, 42 |
| 773 | Mountain-100 Silver, 44 |
| 774 | Mountain-100 Silver, 48 |
| 775 | Mountain-100 Black, 38 |
| 776 | Mountain-100 Black, 42 |
| 777 | Mountain-100 Black, 44 |
| 778 | Mountain-100 Black, 48 |
| 779 | Mountain-200 Silver, 38 |
| 780 | Mountain-200 Silver, 42 |
| 781 | Mountain-200 Silver, 46 |
| 782 | Mountain-200 Black, 38 |
| 783 | Mountain-200 Black, 42 |
| 784 | Mountain-200 Black, 46 |
| 785 | Mountain-300 Black, 38 |
| 786 | Mountain-300 Black, 40 |

For each row included in the results, the existence test evaluated to true. In other words, the returned rows are part of the Mountain Bikes subcategory.

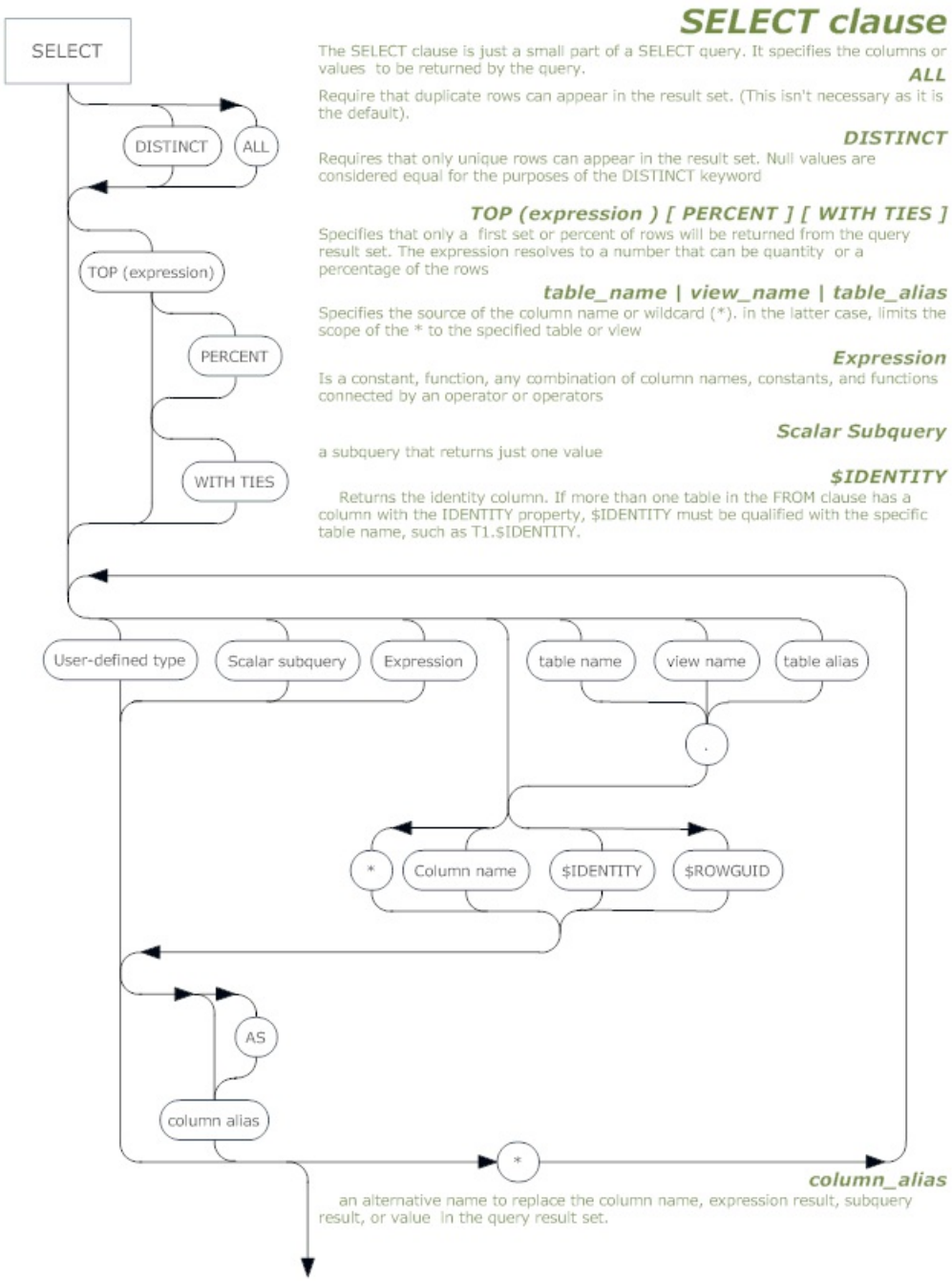
You can also return results for rows whose existence test returns false by using the NOT EXIST operator, as shown in the following example:

```
SELECT
  ProductID,
  Name AS ProductName
FROM
  Production.Product p
WHERE NOT EXISTS
  (
    SELECT *
    FROM Production.ProductSubcategory s
    WHERE p.ProductSubcategoryID = s.ProductSubcategoryID
      AND s.Name = 'Mountain Bikes'
  );
```

Now the statement returns only those rows that are not part of the Mountain Bikes subcategory. Any row whose existence test returns a true is not included in the results.

## Conclusion

As the examples in the article demonstrate, subqueries are a flexible and versatile tool. You can define them wherever an expression is allowed in a SELECT, INSERT, UPDATE, or DELETE statement. You can then use the data returned by the subquery in your outer query to make those statements more powerful and ultimately more useful to your various applications. For more information about subqueries, see the topic “Subquery Fundamentals” as well as other topics that address how to use subqueries in SQL Server Books Online.



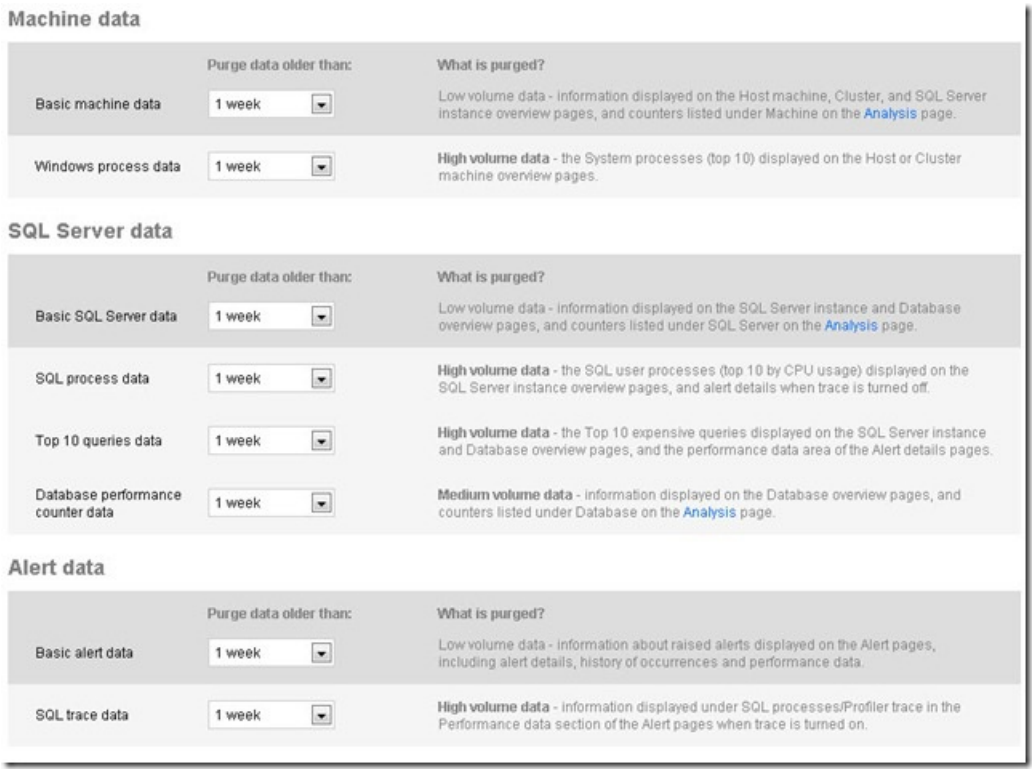

**The Select clause, showing how scalar subqueries can be used within them (A full-size PDF version is available from the speech-bubble at the top of the article**

# Monitoring Baseline

Published Monday, May 30, 2011 3:30 AM

Knowing what's happening on your servers is important, that's monitoring. Knowing what happened on your server is establishing a baseline. You need to do both. I really enjoyed this blog post by Ted Krueger (blog|twitter). It's not enough to know what happened in the last hour or yesterday, you need to compare today to last week, especially if you released software this weekend. You need to compare today to 30 days ago in order to begin to establish future projections. How your data has changed over 30 days is a great indicator how it's going to change for the next 30. No, it's not perfect, but predicting the future is not exactly a science, just ask your local weatherman.

Red Gate's SQL Monitor can show you the last week, the last 30 days, the last year, or all data you've collected (if you choose to keep a year's worth of data or more, please have PLENTY of storage standing by). You have a lot of choice and control here over how much data you store. Here's the configuration window showing how you can set this up:




This is for version 2.3 of SQL Monitor, so if you're running an older version, you might want to update.

The key point is, a baseline simply represents a moment in time in your server. The ability to compare now to then is what you're looking for in order to really have a useful baseline as Ted lays out so well in his post.

by Grant Fritchey
Filed Under: Monitoring

# SQL Server Backup and Restore for the Accidental DBA

24 May 2011
by Grant Fritchey

Not everyone who is tasked with the job of ensuring that databases are backed up, and easily restorable, consider themselves to be database administrators. If you are one of these 'Accidental DBAs' then Grant Fritchey has some good straightforward advice for you to ensure that things go well when a database has to be restored from backups

Here you are. Through some quirk of fate, you're now responsible for your company's databases. People are referring to you as the DBA. You've either volunteered or had the job thrust upon you. Either way, you're expected to know what to do and, let's face it, figuring out where to start is hard. Let me make a suggestion. The very first thing you should check on is the state of your backups. Yes, there are probably a million screaming issues and every single one of them is important, but if your database fails, which they do, and you have no backup…well, let's just say that's not a conversation you want to have with management.

So now you're prepared to get started with backups, but you're unsure of even where to begin. The good news is, this is a well-worn path. Database backups are not a mystery. Again, let me proffer several suggestions.

## Start with the business

Getting your database back online and operational is first and foremost a business decision, not a technical one. Setting up the technical aspects of backup and recovery is relatively easy and extremely well documented, so your initial work needs to be with the business, understanding what you have to deal with there. This serves two purposes. First, you learn what you need to do and just how much infrastructure, planning, and work you have in front of you. Second, you can document what's expected of you, so that if, at some later date, the business questions your technical decisions, you can point back to the discussions you had with them. Here are a set of suggested questions you might want to ask about any given database:

- How important is this database to the company?
- How much time and data can we afford to lose in this database?
- Are there any regulatory requirements regarding keeping backups of the data I need to meet?
- Are there any encryption requirements for data backups?

It's the answers to these questions that begin to drive you down the path to setting up a backup and recovery plan for all the databases that you manage. The business will be able to answer some of these questions quickly and easily. Some they won't. You may have to badger them to get the answers that you need, but you do need to get those answers. Also, you may not be able to work with some answers. For example, the most frequent answer to "How much data can we lose?" is "none." While that may be true, attempting to set up a system with zero chance of data loss is extremely expensive and involves a lot more than simple backup and recovery. Usually you can get the business to agree that a day, or an hour, or 5 minutes would be reasonable expected losses.

Out of all this information you should create a set of documentation describing your backup processes in plain language. You may have a general process that's applicable to all your systems and databases or you might have customized processes for individual databases or applications. Either way, write everything down.

Once you understand the task you have in front of you to satisfy the business, then you can begin to apply technology to the problem. Just remember, as your data expands, grows, and changes, so will the business. Plan on going back to them on a regular basis, at least once a year, to reassess and update your plans as needed.

## Full Backups

All backup plans start with a full backup. A full backup is a low level copy of a database, not only the tables and data, but the indexes, their statistics, all the triggers and stored procedures, and everything that is stored in the database. There is no way for you to arbitrarily pick and choose objects to include in the backup, so the backup will always cover everything, and when you restore the backup, everything will be restored. There are lots of options for a full backup, but here are a few things to remember:

- For recovery to a point in time, the full backup is the base.
- Add CHECKSUM to partially validate backups as they occur.
  NOTE: This will have a noticeable performance impact on larger backup operations.
- You should only backup the database after a consistency check (DBCC CHECKDB).
  NOTE: An exception to this would be if you use the backup itself for a consistency check.
- The full backup does not truncate the log when the database is in FULL or BULK LOGGED recovery modes.

How often you run a full backup is very dependent on the needs of the business. In order to reduce the difficulty and time to recover a backup, for a full backup is recommended at least once a week, but only in conjunction with other backup processes. For small and medium sized databases, up

to approximately 500gb, daily full backups might be a better plan.

Backups can be configured to stack multiple full backups into a single file or they can be configured to have each backup in an individual file. I recommend using individual files, preferably named by date as well as database name. It's easier to see which backups are available and which are missing, if any. If a file becomes corrupt for some reason, you lose a single backup, not all of them.

Just remember, while there are all sorts of other types of backup operations, the foundation of most of them is the full backup. You will need to take that into account on most of your systems for most backup plans.

# Log Backups

Some businesses are fine with only having a full backup and living with the possibility of data loss between whenever a problem occurs and the last full backup. Most businesses are reliant on much more up-to-date data. Because of this, you need to plan for the ability to recover a database to a point in time. This is possible through the use of the FULL recovery mode and a set of log backups. The database must first be set to either FULL or BULK LOGGED recovery. These modes of recovery cause the log to be kept intact until a log backup operation is performed. There are many different options around backing up logs, but a few should be noted:

- In order for the committed transactions to be cleared, a log backup operation must be performed.
- Log backups can use the CHECKSUM operation to partially validate the backup as it occurs.
  NOTE: This will have a noticeable impact on larger log backup operations.

The frequency with which log backups are taken must be determined by the business. Log backups are dependent on having a full backup in place as part of the recovery process. You must run regular log backups in order to truncate the committed transactions within the log. The minimum suggestion would be once per hour. Most businesses operate more in the neighborhood of once every fifteen minutes. I've even managed a system that ran a log backup once every five minutes. Less than that is counter-productive, because the amount of time needed to restore the data will be prohibitive.

Log backups, like full backups, can be configured to all go to a single file or to multiple individual files. Again, I recommend multiple individual files for the same reasons as before.

# Automating the Process

The best way to automate your backup processes is to learn the T-SQL commands and go to work on setting up SQL Agent jobs yourself. Doing this gives you a very high level of control and flexibility. However, that requires a lot of time to develop a good set of knowledge, especially for the accidental DBA. Instead, you can take advantage of built-in automation processes called Maintenance Plans.

The Maintenance Plans have a wizard to walk you through the set-up. It will even schedule things for you. Here are a few notes about using Maintenance Plans:

- Choose "Separate Schedule for each task" on the opening screen of the wizard.
- Do not implement the Shrink Database step.
  NOTE: This is a long and involved discussion. For more, read this article.
- Separate the backups from the other operations in order to minimize resource impact.
- Enable "Verify Backup Integrity".
  NOTE: This is the CHECKSUM option, so it may impact performance.

Once the wizard has created the Maintenance Plan, you can spend a lot of time and effort working with them directly through the GUI interface in SSMS. You have more control over these mechanisms than ever before. But, for extreme fine-grained control you will need to use T-SQL directly and work with SQL Agent to schedule the events.

SQL Agent can not only be scheduled to run any command you want, but it will retry the commands on an error. It can also be programmed to send you emails in the event of a failure. You can work with the scheduled items that you create yourself or even modify those created automatically for you through the Maintenance Plans wizard.

# Full Restore

The most important part of setting up database backups is not the backup operation. The most important part of setting up database backups is being able to restore those backups. This means two things. First, that you verify that the backup files are valid , and can be restored. Second, that you know how to restore these files. Restoring databases is very much like being in a fight. If you practice boxing or some type of martial art you'll know that you practice performing the moves in the air with other people slowly, and with other people at as close to full speed as possible without serious injury. This is all to learn how to move before you have to do it under stress. Restoring a database is exactly the same. There will be plenty of times that you may need to run a restore in a non-stressful situation, but you'll remember the ones when the entire management team is standing in your cube waiting for you to save the company. That is not the time to be searching for a blog post on how to run restores.

Here are a few notes about restoring databases:

- Practice, practice, practice.
- Use the VERIFY ONLY command to partially validate backups.

NOTE: This will only partially validate the file. The only true validation is a restore operation.
- RESTORE FILELISTONLY will allow you to see the definition of files for the database that was backed up.
- RESTORE HEADERONLY will give you information about the backup, other than the files.
- MOVE allows you to create a copy of a database by moving the database files to other locations or other files.

Just remember that the full restore is not selective. You will get everything that was in the backup. If a particular row, table, or procedure is missing, that's because it wasn't in the backup.

# Point In Time Restore

Because data may have changed since the last full backup, you configure your database to support log backups. When the time comes to restore the logs, you can restore to a specific point in time. Even more practice is required with the point in time recovery because so many more steps are involved. Here are some key points to remember:

- Practice, practice, practice.
- You must first restore a full backup, but it must be left in the Recovering state.
- Until you reach the final log backup, each section of the log that is restored must be left in a Recovery state.
- You must restore the logs in the order in which they were taken. You cannot skip sections of time.

# Compression

One of the most important additions in recent years is the ability to compress backups. Compression helps conserve disk space but more importantly, it sacrifices some memory and CPU cycles to actually arrive at faster backup execution speeds. This is because less data is written to the disk and disks are the slowest part of the system. Because of this, unless your system is suffering extreme CPU or memory loads, it's worth using compression on all of your backup operations.

The only problem with this is that backup compression is only available for certain versions of SQL Server. Compression was introduced with SQL Server 2008 Enterprise. When Service Pack #2 was released, they made compression available for SQL Server 2008 Standard. SQL Server 2008 R2 has the same restrictions, compression on Enterprise and Standard only. The Development version, since it's effectively the Enterprise version, also has compression available. No other versions of SQL Server can create compressed backups or restore compressed backups.

# Additional Backup Considerations

This introduction only covers the bare bones of standard backup and restore operations. There are an extremely large number of additional topics that you can add to the mix. This is a small sampling of the more important aspects:

## Differential Backups

Full backups can take a lot of time, and log backups only involved transactions that have yet to be backed up. Differential backups provide a middle ground between the two. A differential backup retrieves all the committed data that has changed since the last full backup, and creates it in a backup, representing the difference between the current moment and the last full backup. Differential backups can then be used in restore operations to either bring the database up to the moment of the differential, like applying a log backup, or in conjunction with log backups. Like log backups, the full backup must be left in a recovering state in order to apply differential backups. For larger systems, these backup types can add additional flexibility to your disaster recovery planning.

## File and File Group Backups

While it's not possible to backup individual pieces of a database, such as just one table, there is a way to sort of get around this. It is possible to back up each file or file group independently of the others, either through a full or differential backup. These backups will back up everything that is on a given file or filegroup. You can then restore just the file or filegroup to the database. If only a single table is stored on that file group, then you can restore just that table. There are a number of restrictions around it, but again, for larger systems, this provides additional flexibility and power for planning your backup schedules.

## Snapshot

Snapshot backups are not so much a part of disaster recovery as they are a mechanism of safety during specialized operations on the database such as deployments. Essentially a snapshot creates a moment in time copy of the database, very quickly. You can then, just as quickly, restore the snapshot to the database. These are not traditionally used in disaster recovery scenarios since the snapshot only contains changes made to the database. There is no way a snapshot can be restored unless the underlying database is already in place, the opposite of a disaster recovery scenario.

## Copy Only

Since a full backup is the basis on which point in time recovery is built, whenever you take a full backup you introduce a new starting point for all the log and differential backups. Sometimes you might just want to take a backup of the database without affecting this chain. Using the COPY ONLY

command allows you to do this.

### Striping

You are not forced to write your backups to a single file on a single disk drive. You can set them up to stripe across multiple files on multiple drives. This is a way for larger systems to get a full backup in place and a mechanism for somewhat, but not drastically, faster backups.

### Encryption

Most businesses require you to password-protect your systems. It's also possible they may require you encrypt backups. To do this using SQL Server you have to encrypt the database, but then the backups will be encrypted as well.

## Conclusion

Backups are a very important part of the DBA's duties, but the most important part of backups is actually the restore. You need to know that you have good backups in place and that you can restore them. You must also work with the business in order to arrive at a good backup plan. Once you have it in place, make sure you validate your backups to ensure they work, because it is all on your shoulders. You're the DBA.

## The Checklist

- Full backups running on a regular basis.
- Full backup to individual files.
- Use CHECKSUM with full backup.
- DBCC run as part of full backup processing.
- When database is in FULL or BULK LOGGED recovery, run log backups.
- Log backup to individual files.
- Use CHECKSUM with log backup.
- Minimum frequency on log backups is one per hour.
- Automate the backup process.
- Practice restore operations frequently.
- Validate backups by restoring database.
- Use VERIFYONLY for partial validation of backups.
- Use backup compression when available.
- Document backup and restore processes.

# Monopolytics: Porting the .NET framework

Published Friday, May 20, 2011 12:29 PM

.NET was originally conceived as a portable framework that would run on any number of platforms. Microsoft has gradually diminished their ambitions for .NET and Silverlight, but as long as Mono and Moonlight lived, there was proof that the framework can be ported. It came as a surprise to many, therefore, that Attachmate should close down the Mono project so soon after purchasing Novell, and fire all the Mono team. Surely a Linux port of the .NET framework is something that the industry wants, and what Microsoft needs?

It might seem odd to worry about .NET's credibility whilst it continues to dominate the industry. However, Microsoft's poor showing with windows-based mobile devices and tablets means that a .NET framework that can only run on Windows isn't an option. .NET developers need Mono, and the commercial products that use it such as MonoTouch, Unity3D, and MonoDevelop to give them the confidence to develop applications for the whole range of successful mobile devices and servers.

Fortunately for the .NET community, the founder of the Mono project, Miguel de Icaza, announced a fortnight after the Attachmate putch that the project lived, with the mono team working for a new company Xamarin. It looks as if they had been planning to create a spin-off in time but not expecting the layoffs. This means that there will now be three companies producing commercial products based on Mono, Grasshopper for ASP.NET on Linux servers, Unity with an iOS framework and Xamarin with iOS, and Android offerings.

A managed framework is a great help for developing iOS applications even if it is just to allow you to forget about memory management. Added to that is the huge resources of the .NET framework. A programmer can develop for iOS, Android and Windows and share almost all the non-GUI code. It could even be used for a Windows Phone 7 version! If current plans succeed, we might even eventually be able to port GUI code using a strong Mono Silverlight implementation on iOS and Android if they can get over the poor fit of the Silverlight/Moonlight Binary Interface with android or iOS.

Now that there seems to be a reasonable long term strategy for Microsoft in the mobile market in combination with Nokia and Skype, It would be great to get this final missing part of the story in place.

Cheers,

Laila

by Laila

# Be the Puppet Master! Control Multiple PCs with one Keyboard and Mouse

20 May 2011
by Wesley David

> In the average IT department, geek-credibility is bound up with the number of monitors you are simultaneously using. What about going one step further, and running them on several computers at once, with just one mouse and keyboard?

Previously, I discussed how to extend your desktop across another computer's monitor. This would be useful if you have no more video ports on your main PC but still have a few spare monitors on which you'd like to spread your workspace across. It's also useful in repurposing an old laptop that you haven't gotten around to properly disposing of.

Notwithstanding the above, you may find yourself in the exact opposite position; that of wanting to use your main keyboard and mouse to spread out across several independently functioning PCs. This is usually done with a physical KVM (Keyboard, Video, Mouse) switch that allows you to switch one keyboard, monitor and mouse between different computers at the mere click of a button. An example would be this CablesToGo TruLink DVI KVM switch. However, what if the secondary computers involved have their own independent monitors that you want to use? So instead of a KVM solution it's more of a "KM" solution. Keep that in mind. This software solution does not forego the need for each computer to have its own monitor.

So what's not to like about KVM switches? They're lovely! "KVM switches walk in beauty, like the night. Of cloudless climes and…" Oh, but wait one moment thou budding Byron! A physical KVM switch can be a lordly pest in two ways. The first is that it costs money. Oh sure, for some departments and businesses a $100 to $400 investment is acceptable, especially for such a star member of the IT team as yourself. However in some places the denarii are kept under the watchful guard of a fierce cohort headed by the most dread centurion of all, the CFO. Any solution that is either free or involving the least possible expense is always appreciated in those environments.

A second limitation of physical KVM switches is that switching from controlling one PC to another comes at the expense of visual contact with the first PC. You can only be viewing one PC at a time since by necessity the KVM switch is connected to one main monitor (or pair of monitors if you have a fancy switch). So you may have a query window open on one PC and then need to switch to the resource monitor on another PC, but you can't view both at the same time.

Cue the entrance of software-based KVM solutions! Of course, keep in mind that they're more properly known as "KM" software. KM software does not transport the remote computer's video to you, it merely transports your keyboard and mouse input to the remote machine (hence the missing "V" in the acronym) and relies on the remote machine's ability to drive video to its own monitor.

Having said that, it's easy to see that the following utilities only work in a somewhat peculiar set of circumstances. To reiterate, you need to have multiple PCs, each with their own monitor(s) all within a close physical proximity of each other. If you only have one monitor to share between multiple PCs, then you're stuck pleading with your centurion for some financing and hoping that he doesn't retaliate and unleash his fiercest legionaries to audit your department's budget. Do not mock the pasty appearance of an accountant. They can do things with profit and loss statements that can cripple the heartiest among us.

There are a number of utilities to perform the task laid out above. For the sake of this article I'll focus on two main titles. The first is a commercial product, and the second is FOSS (That's "Free Open Source Software" for those of us who spend too much time around proprietary kit).

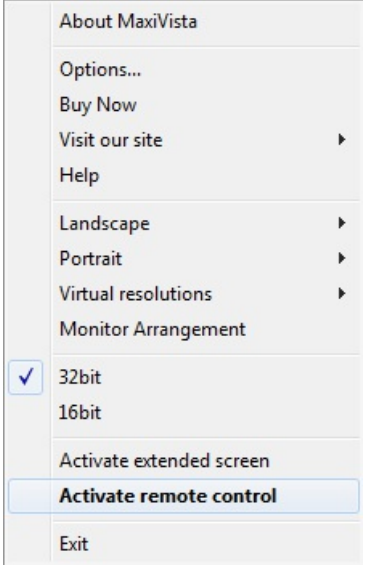
## MaxiVista

Does this software sound familiar? If you read the sister article to this one titled "Monitors! Windows Extended Across Windows" you'd recall that MaxiVista was one of the tools that allowed us to utilize another PC's monitor as an extension to our main PC. Quite a versatile tool, MaxiVista also allows us to interact with multiple PCs using one keyboard and mouse.

MaxiVista comes in several editions, the cheapest one merely allowing you to have an extended screen across another PC's monitor. If you'd like the ability to use MaxiVista as a KM tool, you'll need to purchase at least the mezzo edition for $10 more. Fortunately you can download a trial to see if the product works for you.

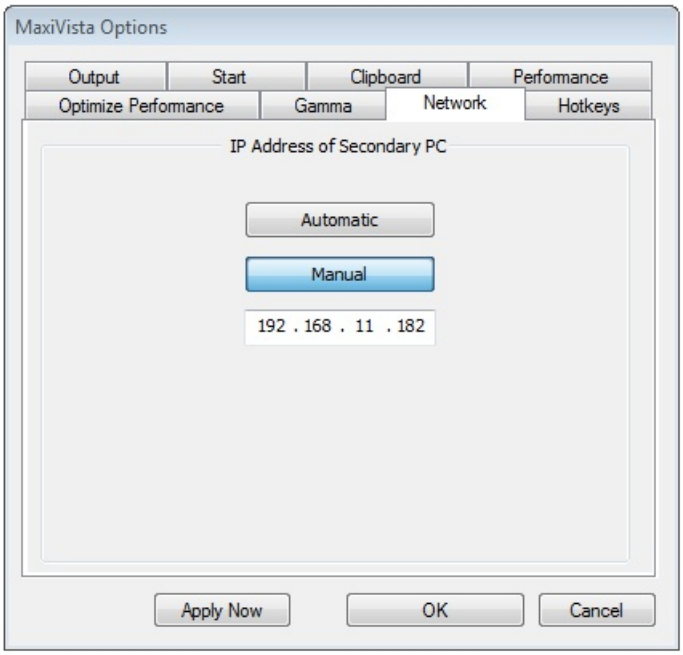
One of the glaring limitations of MaxiVista is that it only works on Microsoft Windows. For many, I'm sure that is an acceptable limitation. However, some of us have a hectic gamut of operating system polytonality humming in our cubicle. If you can live with a Microsoft-centric solution, then continue reading. If not, skip down to the section on a competing tool called Synergy.

The installation of MaxiVista is straightforward enough. Very few options present themselves during installation. There are two installers included with MaxiVista. One of them is for the primary or "server" PC. The other is for the secondary PC(s) that you'd like to be able to control in addition to your main PC. As per the previously mentioned article, they could also be the PCs that you'd like to extend the primary PC's desktop onto. It's worth noting that it's as simple as a right click to go from a KM setup to an extended desktop one.




This makes it incredibly easy to be controlling the PC next to you one minute and then simply choose to extend your desktop across the secondary PC the next. Quite a handy trick you've kicked off long-running operations on your secondary PC but you still want the extra screen real estate. Talk about multi-purposing your equipment!

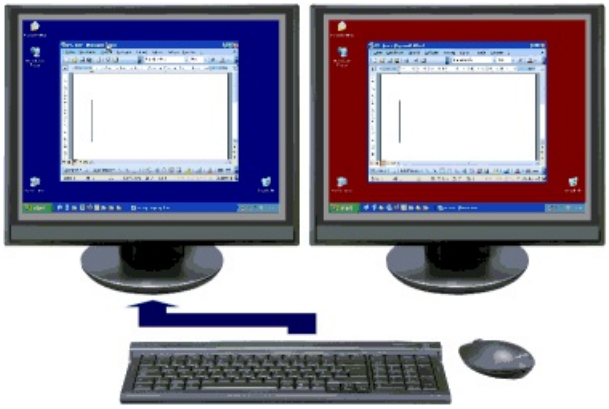
Once you've installed the server and client components on the PCs involved, you'll need to edit the options on the "server" or main PC. Right click the system tray icon for MaxiVista and select "options". Lots of options exist and many of them are left as an exercise for the reader (they're fairly self-explanatory; promise!). The options of primary importance for us right now are on the "network" tab.




On the network tab you can choose to scan the local subnet for devices that are running the client MaxiVista software or you can manually enter the IP addresses (helpful if your clients are on a separate subnet). In my case you only see the option to add a single secondary computer because I'm using the trial version of the software which is limited to only one secondary PC.

Your next task will be to open your main PC's display properties and arrange the MaxiVista virtual monitor. In my case, MaxiVista's virtual monitor is display 3. I've arranged it on the right of my main display since that's the physical location of the monitor connected to my secondary PC.


Once that bit of housekeeping is finished I can now seamlessly move my mouse and keyboard across two PCs. This is an animated gif that illustrates how MaxiVista will work in "remote control mode".



It should also be noted that MaxiVista is limited to installing only four virtual display drivers so you can only ever have up to four extended or remotely controlled displays. That includes secondary PCs with multiple monitors. If perchance you have three secondary PCs, each with two monitors, you'll only be able to utilize two of those PCs since they combine for a total of 4 monitors. One PC with four monitors will max out your quota as well.

But what if you have multiple non-Windows machines that you'd like to remotely control? Or perhaps your ice-blooded centurion won't even budge on the $49 USD price tag of MaxiVista's middle tiered edition. What is a poor plebeian to do? FOSS to the rescue!

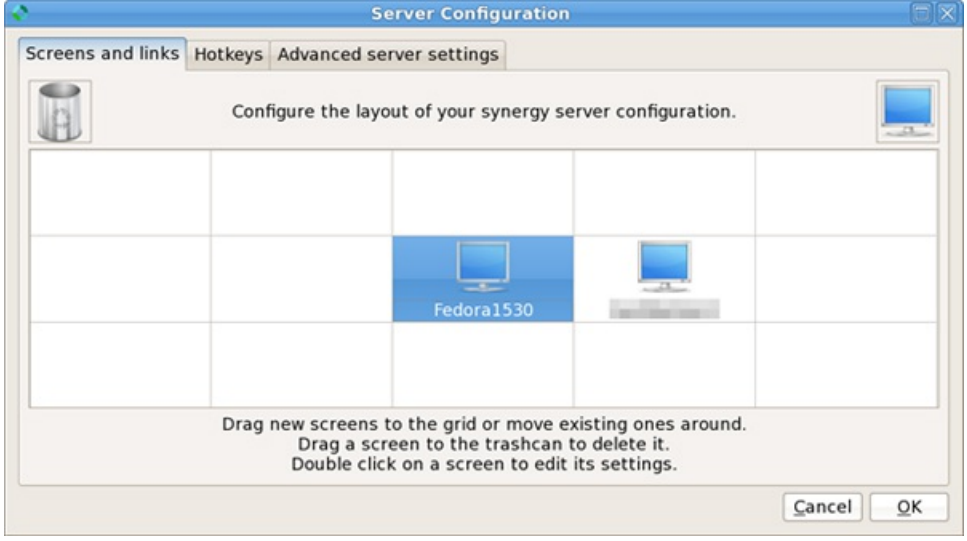
## Synergy

This project started as Synergy, but then fell silent in 2006. It was forked as Synergy+ in 2009, however in more recent times Synergy and Synergy+ have been merged. The project has now gone back to its original name (no more "+" symbol) and also has a snazzy Redmine-based website. Synergy strives to make a tool that allows for the seamless transition of a user's keyboard and mouse to another PC on virtually any modern operating system. With packages spanning the OS spectrum from Windows to OS X to Ubuntu / Debian to Fedora / RedHat (including both 32 and 64 bit for each), there aren't many people who will be left out of the party.

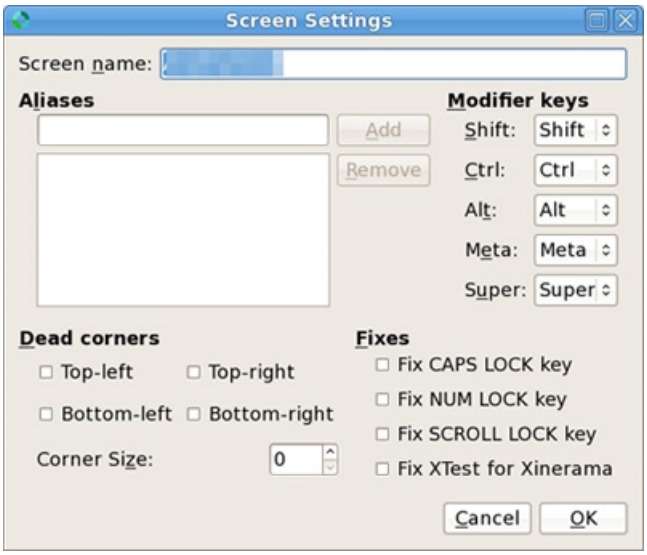

**Graphic property of its respective owners. Originally found on http://synergy-foss.org/**

There is no software separation of the client and server roles. When installed, the Synergy app can act as the client or server. Setting up the server is friendly enough in the latest 1.4.2 version (it used to be a lot worse, trust me).

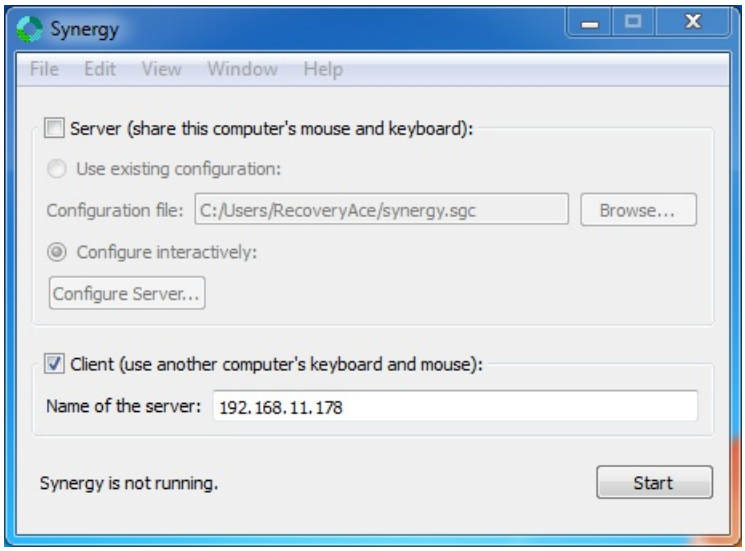

To get up and running with Synergy, you'll need to configure the server just a little bit. On the server configuration screen, you'll see a matrix of cells that can contain different screens. By dragging and dropping the monitor located in the upper right of the dialog box, you can add remote screens and determine which side of your main PC's screen will be the gateway to your secondary PCs.




Take careful note though! When editing the options for those secondary screens, you must name them exactly the name that those computers are actually given. Not just any remote PC can connect to a Synergy server; only those that the server is expecting via their name. The input box titled “screen name” is what corresponds to the machine’s name. You can also make multiple aliases for a single machine, however that isn’t necessary.




Once the server is expecting a client, you can start the service on the client PC and input the server’s IP address or DNS name and click “start”.




If you have trouble connection your clients to your server, make sure that the server has TCP port 24800 unblocked. As you might have noticed from the pictures above, I’m running the Synergy server on a Linux machine (Fedora 14 to be precise) and the client on a Windows machine (Windows 7 Professional x64 ). I can happily sachet between PCs using the same keyboard and mouse. Today, for instance, I would take a few moments to check up on the computations the Windows 7 machine was performing and then effortlessly move back to my Fedora machine to get back to more menial tasks.

Concerned about security? Since Synergy does not have any encryption or authentication in place in the current version (with the exception of the limited “authentication” provided by the PC name restriction mentioned above) you must provide your own. If you want an encrypted connection

between PCs, you can simply run an SSH server on each PC encrypt traffic on port 24800.

Synergy does provide some more exciting options, such as the ability to lock your cursor to one remote screen (think “gaming”), the ability to autostart the program, custom hot keys to quickly switch the cursor to a different screen, and more!

Is Synergy perfect? Well, no. As I use it, sometimes the cursor will disappear and I’ll be forced to blindly struggle for a bit before it reappears. However, I think that has more to do with me using some virtualization gizmos on my Fedora machines than it does a pure Synergy bug. Encouragingly, the project is alive and healthy with plenty of activity on their Redmine site logged virtually every day.

Between Maxivista and Synergy, there should be a solution to make you a little happier as well as appease the mighty centurion in the corner office with the “Trust me, I’m a comptroller” coffee mug. If not, I hear rowing in the galleys will give you a killer set of latissimus dorsi so you’ve got a career as a fitness model ahead of you!