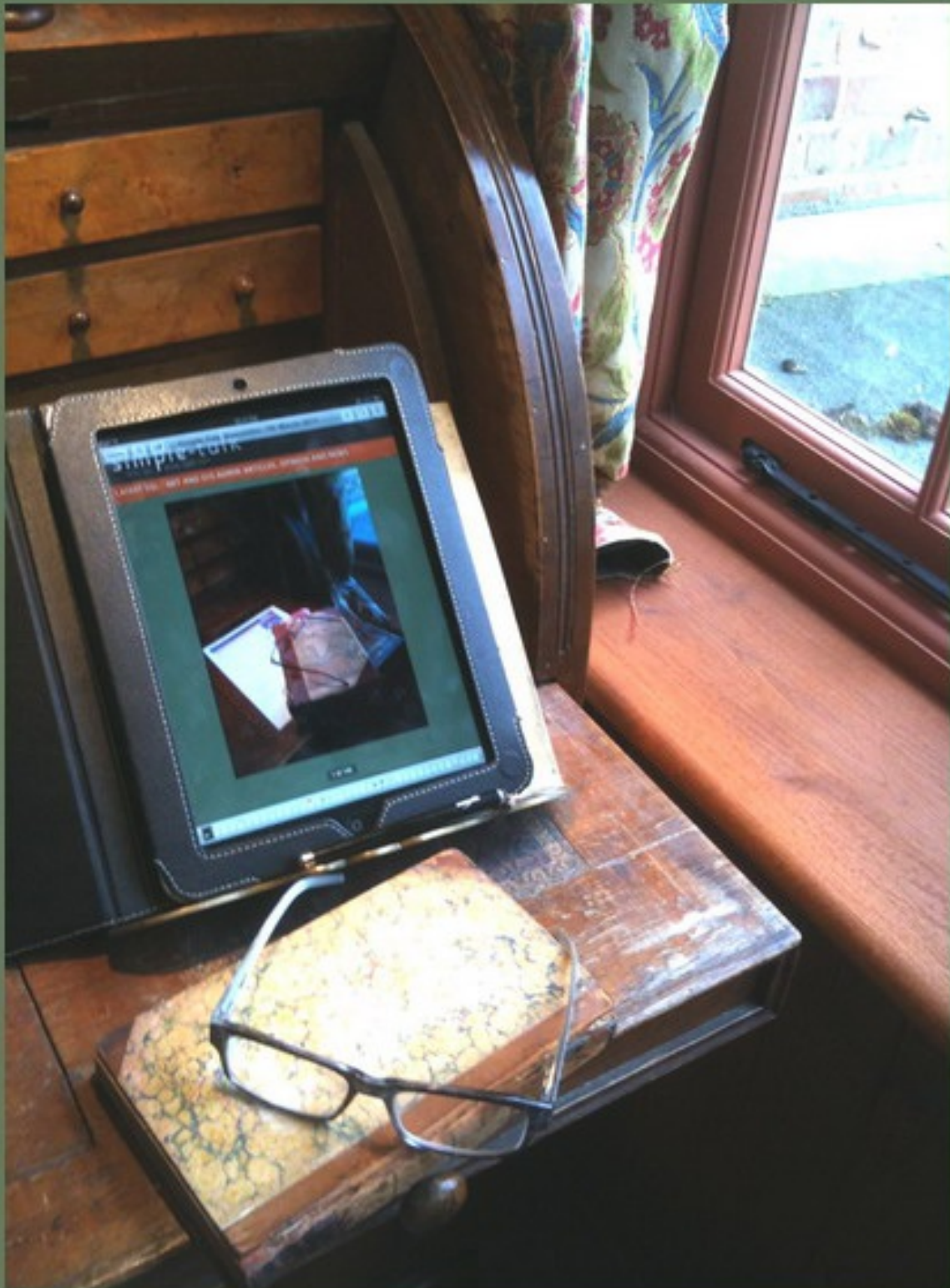


# simple-talk

DATA, DEV, ADMIN, CHOP SUEY

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



# Normalisation and 'Anima notitia copia' (Soul of the Database)

Published Friday, February 17, 2012 6:23 AM

(A Guest Editorial for Simple-Talk)

The other day, I was staring at the **sys.syslanguages** table in SQL Server with slightly-raised eyebrows .

da...	d..	u...	name	alias	months
mdy	7	0	us_english	English	January,February,March,April,May,June,July,August,September,Octobe...
dmy	1	0	Deutsch	German	Januar,Februar,März,April,Mai,Juni,Juli,August,September,Oktober,Nov...
dmy	1	0	Français	French	janvier,février,mars,avril,mai,juin,juillet,août,septembre,octobre,novembr...
ymd	7	0	日本語	Japanese	01,02,03,04,05,06,07,08,09,10,11,12
dmy	1	0	Dansk	Danish	januar,februar,marts,april,maj,juni,juli,august,september,oktober,novemb...
dmy	1	0	Español	Spanish	Enero,Febrero,Marzo,Abril,Mayo,Junio,Julio,Agosto,Septiembre,Octubre...
dmy	1	0	Italiano	Italian	gennaio,febbraio,marzo,aprile,maggio,giugno,luglio,agosto,settembre,ott...
dmy	1	0	Nederlands	Dutch	januari,februari,maart,april,mei,juni,juli,augustus,september,oktober,nove...
dmy	1	0	Norsk	Norwegian	januar,februar,mars,april,mai,juni,juli,august,september,oktober,novemb...
dmy	7	0	Português	Portuguese	janeiro,fevereiro,março,abril,maio,junho,julho,agosto,setembro,outubro,n...
dmy	1	0	Suomi	Finnish	tammikuuta,helmikuuta,maaliskuuta,huhtikuuta,toukokuuta,kesäkuuta,...
ymd	1	0	Svenska	Swedish	januari,februari,mars,april,maj,juni,juli,augusti,september,oktober,novem...
dmy	1	0	čeština	Czech	leden,únor,březen,duben,květen,červen,červenec,srpen,září,říjen,listo...
ymd	1	0	magyar	Hungarian	január,február,március,április,május,június,július,augusztus,szeptember,o...
dmy	1	0	polski	Polish	styczeń,luty,marzec,kwiecień,mai,czerwiec,lipiec,sierpień,wrzesień,paździ...

I'd just been reading Chris Date's interesting book 'SQL and Relational Theory'. He'd made the point that you're not necessarily doing relational database operations by using a SQL Database product. The same general point was recently made by Dino Esposito [about ASP.NET MVC](#). The use of ASP.NET MVC doesn't guarantee you a good application design: It merely makes it possible to test it. The way I'd describe the sentiment in both cases is 'you can hit someone over the head with a frying-pan but you can't call it cooking'.

SQL enables you to create relational databases. However, even if it smells bad, it is no crime to do hideously un-relational things with a SQL Database just so long as it's necessary and you can tell the difference; not only that but also only if you're aware of the risks and implications. Naturally, I've never knowingly created a database that Codd would have frowned at, but around the edges are interfaces and data feeds I've written that have caused hissy fits amongst the Normalisation fundamentalists. Part of the problem for those who agonise about such things is the misinterpretation of Atomicity. An atomic value is one for which, in the strange virtual universe you are creating in your database, you don't have any interest in any of its component parts. If you aren't interested in the electrons, neutrinos, muons, or taus, then an atom is ..er.. atomic. In the same way, if you are passed a JSON string or XML, and required to store it in a database, then all you need to do is to ask yourself, in your role as *Anima notitia copia* (Soul of the database) 'have I any interest in the contents of this item of information?'. If the answer is 'No!', or 'nequequam!' Then it is an atomic value, however complex it may be. After all, you would never have the urge to store the pixels of images individually, under the misguided idea that these are the atomic values would you? I would, of course, ask the 'Anima notitia copia' rather than the application developers, since there may be more than one application, and the applications developers may be designing the application in the absence of full domain knowledge, ('or by the seat of the pants' as the technical term used to be). If, on the other hand, the answer is 'sure, and we want to index the XML column', then we may be in for some heavy XML-shredding sessions to get to store the 'atomic' values and ensure future harmony as the application develops.

I went back to looking at the **sys.syslanguages** table. It has a **months** column with the months in a delimited list

January,February,March,April,May,June,July,August,September,October,November,December

This is an ordered list. Wicked? I seem to remember that this value, like **shortmonths** and **days**, is treated as a 'thing'. It is merely passed off to an external C++ routine in order to format a date in a particular language, and never accessed directly within the database. As far as the database is concerned, it is an atomic value. There is more to normalisation than meets the eye.

by [Phil Factor](#)

# Automated Script-generation with Powershell and SMO

03 February 2012

by Phil Factor

In the first of a series of articles on automating the process of building, modifying and copying SQL Server databases, Phil Factor demonstrates how one can generate TSQL scripts for databases, selected database objects, or table contents from PowerShell and SMO.

If you're a DBA or a database developer, you've probably found that point'n'click script-generation via SSMS becomes an increasingly tedious chore, and one that always increases the chance of error. Now, if you're happy with the tedium of using SSMS to create your database scripts regularly, then use it, because it works, and it is probably the best way of getting one-off ad-hoc scripts for special purposes.

In my case, however, I want automated scripting, because it gives me the opportunity of getting the result exactly as I want it. I'm then more confident of having an up-to-date database build-script to hand, and I'm more likely to use this technique to perform tedious routine jobs such as integration, source control and creating test databases.

A live database is created from a number of SQL scripts, both DDL and DML. Script-generation is the act of reverse-engineering that live database, or a database component such as a schema or set of routines, into one or more scripts that are capable of creating an exact replica.

You'll probably want a complete build script, as it is difficult to create a build script for a working database from individual object scripts alone, and impossible unless you are aware of the server settings and have scripted the database settings too. It is therefore wise to regularly save a complete script for building the entire database, and maybe a separate script for the database settings as well. You'll probably also want to use script-generation to get the source of individual routines, tables and views for source control, for editing and for your own records.

There are a number of choices to be made when creating your database scripts, depending on the problem you're trying to solve. It's like buying a frou-frou cup of coffee. You are bombarded with decisions. Are you updating the schema of an existing database, or are you creating it from scratch? Do you only wish to update the routines? Do you want to temporarily disable the constraints in order to import data in the 'wrong' order? Do you want to leave out the DRI until you've imported all the data?

You need this flexibility required in a script-generation solution because you have a fair amount of complexity to deal with. A database application, at any point in time, consists of one or more databases on one or more servers, and may have more than one version or fork being developed. They may be using different data. As well as the code for all the routines, views and tables, there will be database settings and server settings. Finally, there will be data, even if just the basic enumerations and static data without which nothing works. As well as the code in the shared 'public' database, you may also have stealth things you are trying out, and sandbox stuff that needs to be preserved. You will also need to script your endpoint configuration and tasks that will go on the SQL Server agent. If you're doing serious website work, you'll have queues managed by service broker too.

If the production-DBAs have scripts for all of this, for all current versions, along with migration and rollback scripts, then they are smiling.

As well as squirreling away the code in order to preserve the work of the team, keep track of progress, maintain test-cells, do builds and rollbacks from builds, and to relate bugs to code alterations, you need codes to understand the database. You can understand a minnow of a database such as Adventureworks through Point n' click in SSMS, but for an industrial-strength behemoth, then it is far quicker to eyeball down the build scripts. I know of very few ways to generate database scripts, and a lot of these do it wrong, because the scripts are formatted as a machine-to-machine communication. Table build scripts, for example, can be written for legibility or merely in order to get a correct build. In order to quickly learn a database, you need the legible version.

No, I had to write my own version and it has paid dividends. Functions, for example, are easier to understand with a structured comment block listing the parameters and comments in extended properties, and even where they are referenced and what they reference. Tables are far better with the simpler constraints written in-line, and comments on both table and column pulled from the extended properties.

## Automated scripting of database DDL

Let's start with the obvious technique. We'll use PowerShell and Server Management Objects (SMO). No, don't panic. SMO is a monster, but that's because it is written for many purposes, the worst of which is providing an interface between SSMS and the servers. In most cases, there is already a high-level interface for the likes of you and me. I'll admit that SMO isn't easy. It is always a bad sign when you've got a problem with a script, and you reach for Google, only to find nothing more than a string of Chinese characters on half a page, and a similar StackOverflow question left unanswered since 2008, save for advice to use SSIS instead.

This sort of thing tends to happen when you're using SMO, which is such a shame because it is so powerful. In fact, almost any job that you can do via SSMS you can do through SMO. With PowerShell and SMO, you can work magic, but with that strange, lonesome feeling that not many people have ever walked down the same path. It's the awful documentation, combined with the intricate, arcane, multi-layered interface that makes SMO hard work for the uninitiated, but it shouldn't be like this.

Microsoft has always shown its ambivalence in letting users loose on SMO, by neglecting to provide anything remotely resembling adequate documentation. All we get is minimal, and probably machine-generated, documentation of the SMO classes, methods and so on. Even the examples have errors. Microsoft has to keep SMO up to date because it is used for SSMS, but there seems to be a great deal of passive

resistance to supporting users who need to use it for scripting. For this reason, I'll be keeping these examples as simple as I can.

## Getting stuck in quickly

There is usually a quick way to do things. Here is the PowerShell to script out the MyDatabase database from MyServer into the local directory E:\MyScriptsDirectory' (it will do others, of course, by changing the assignments to the three variables at the head of the script). Note that in these PowerShell scripts I've opted to avoid the terse style, mainly because the terse style is less intelligible for those of us who just want to use PowerShell without getting too absorbed.

```
$DataSource='MyServer'
$Database='MyDatabase'
$FilePath='E:\MyScriptsDirectory'
$Filename= $FilePath+'\'+$Database+'.sql'
# Load SMO assembly, and if we're running SQL 2008 DLLs Load the SMOExtended and SQLWMIManagement Libraries
$v = [System.Reflection.Assembly]::LoadWithPartialName( 'Microsoft.SqlServer.SMO')
if (((($v.FullName.Split(',') [1].Split('=') [1].Split('.') [0] -ne '9') {
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SMOExtended') | out-null
}
}$My='Microsoft.SqlServer.Management.Smo'
$s = new-object ("$My.Server") $DataSource
$CreationScriptOptions = new-object ("$My.ScriptingOptions")
$CreationScriptOptions.ScriptBatchTerminator = $true # this only goes to the file
$CreationScriptOptions.Filename = $filename;
$CreationScriptOptions.ToFileOnly = $true # no need of string output as well
$transfer = new-object ("$My.Transfer") $s.Databases[$Database]
$transfer.options=$CreationScriptOptions # tell the transfer object of our preferences
$transfer.EnumScriptTransfer()
"All done"
```

I reckon this is the simplest PowerShell script to get an executable build script, and it isn't too painful. Die-hard SMO-heads will notice that I have to write to a file via SMO in order to get the batch-terminator GO into the script. To do this, I've created a 'ScriptOptions' object, which isn't entirely necessary yet, but will be once we increase the complexity of the task. If you run the script, it will successfully build a database, but there will be a lot missing, because we've been using the default options for generating the script. You'll have no DRI. The tables, in other words, won't have their constraints and indexes, or any dependent objects at all. It will miss out all the extended properties as well.

There is some work to be done. Not all the defaults for the script options are sensible. A quick bit of PowerShell to query the ScriptOptions object will tell us what the defaults are. I'll print them all out because this is a useful reference when you're struggling with a script task. This is the equivalent of the options for the frou-frou cup of coffee, and you've just been served 'black without sugar'. Our scripting options are below, along with the defaults.

## Options: 'Do you want cinnamon with that coffee, sir?'

These are the scripting options with their default settings. I haven't explained them, because they are either obvious or undocumented

FileName	
Encoding	System.Text.UnicodeEncoding
DirWithNoCheck	False
IncludeFullTextCatalogRootPath	False
BatchSize	1
ScriptDrops	False
TargetServerVersion	Version110
TargetDatabaseEngineType	Standalone
AnsiFile	False
AppendToFile	False
ToFileOnly	False
SchemaQualify	True
IncludeHeaders	False
IncludeIfNotExists	False
WithDependencies	False
DirPrimaryKey	False
DirForeignKeys	False
DirUniqueKeys	False
DirClustered	False
DirNonClustered	False
DirChecks	False
DirDefaults	False
Triggers	False
Statistics	False



ClusteredIndexes	False
NonClusteredIndexes	False
NoAssemblies	False
PrimaryObject	True
Default	True
XrnlIndexes	False
FullTextCatalogs	False
FullTextIndexes	False
FullTextStopLists	False
Indexes	False
DrIIndexes	False
DrIAllKeys	False
DrIAllConstraints	False
DrIAll	False
Bindings	False
NoFileGroup	False
NoFileStream	False
NoFileStreamColumn	False
NoCollation	False
ContinueScriptingOnError	False
IncludeDatabaseRoleMemberships	False
Permissions	False
AllowSystemObjects	True
NoIdentities	False
ConvertUserDefinedDataTypesToBaseType	False
TimestampToBinary	False
AnsiPadding	False
ExtendedProperties	False
DdlHeaderOnly	False
DdlBodyOnly	False
NoViewColumns	False
SchemaQualifyForeignKeysReferences	False
AgentAlertJob	False
AgentJobId	True
AgentNotify	False
LoginSid	False
NoCommandTerminator	False
NoIndexPartitioningSchemes	False
NoTablePartitioningSchemes	False
IncludeDatabaseContext	False
NoXrnlNamespaces	False
DrIIncludeSystemNames	False
OptimizerData	False
NoExecuteAs	False
EnforceScriptingOptions	False
NoMailProfileAccounts	False
NoMailProfilePrincipals	False
NoVardecimal	True
ChangeTracking	False
ScriptDataCompression	True
ScriptSchema	True
ScriptData	False
ScriptBatchTerminator	False
ScriptOwner	False

This is a scarily complicated set of options. One can't help wondering why some of these options would be required. However, we can soon put our script right without too much bother. Your own requirements may be different, but I was aiming for an exact copy of AdventureWorks in my testing!

## Making an exact copy of AdventureWorks

```

$DataSource='MyServer
$Database='MyDatabase'
$FilePath='E:\MyScriptsDirectory'
$Filename= $FilePath+'\'+'+$Database+'.sql'
# Load SMO assembly, and if we're running SQL 2008 DLLs load the SMOExtended and SQLWMIManagement Libraries
$v = [System.Reflection.Assembly]::LoadWithPartialName( 'Microsoft.SqlServer.SMO')
if (((($v.FullName.Split(',') [1].Split('=') [1].Split('.')[0] -ne '9')) {
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SMOExtended') | out-null
}
$My= 'Microsoft.SqlServer.Management.Smo'

```

```

$s = new-object ("My.Server") $DataSource
$CreationScriptOptions = new-object ("My.ScriptingOptions")
$CreationScriptOptions.ExtendedProperties= $true # yes, we want these
$CreationScriptOptions.DRIAll= $true # and all the constraints
$CreationScriptOptions.Indexes= $true # Yup, these would be nice
$CreationScriptOptions.Triggers= $true # This should be included when scripting a database
$CreationScriptOptions.ScriptBatchTerminator = $true # this only goes to the file
$CreationScriptOptions.IncludeHeaders = $true; # of course
$CreationScriptOptions.ToFileOnly = $true #no need of string output as well
$CreationScriptOptions.IncludeIfExists = $true # not necessary but it means the script can be more
versatile
$CreationScriptOptions.Filename = $filename;
$transfer = new-object ("My.Transfer") $s.Databases[$Database]
$transfer.options=$CreationScriptOptions # tell the transfer object of our preferences
$transfer.EnumScriptTransfer()
"All done"

```

If you test a database built with this script against the original, it gives a pretty good account of itself. All that's missing are some extended properties on indexes, but there is no switch that one can flip to tickle those out of SMO, so I suspect that someone has made a mistake.

We did this intermediate version because it is simple and demonstrates a clean technique which you can take and expand on. It's great for archiving a complete build script that you can use in source control alongside the individual object scripts. However, I'm going to include a more complete version that will give you a database build script and an object-deletion script as well as the object-build script, all concatenated into one script. You'll begin to understand why I like to create a 'ScriptingOptions' object to store the options, since it is more efficient for this sort of job.

## Getting database settings and object drops into a database-script

```

# set "Option Explicit" to catch subtle errors
set-psdebug -strict
$DirectoryToSaveTo='e:\MyScriptsDirectory\' # Local directory to save build-scripts to
$servername='MyServer # servername and instance
$Database='MyDatabase' # the database to copy from
$ErrorActionPreference = "stop" # you can opt to stagger on, bleeding, if an error occurs
Trap {
# Handle the error
$err = $_.Exception
write-host $err.Message
while( $err.InnerException ) {
    $err = $err.InnerException
    write-host $err.Message
};
# End the script.
break
}
# Load SMO assembly, and if we're running SQL 2008 DLLs Load the SMOExtended and SQLWMIManagement Libraries
$v = [System.Reflection.Assembly]::LoadWithPartialName( 'Microsoft.SqlServer.SMO')
if (((($v.FullName.Split(',') [1].Split('=') [1].Split('.') [0] -ne '9') {
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SMOExtended') | out-null
}
)$My='Microsoft.SqlServer.Management.Smo'
$s = new-object ("My.Server") $ServerName # get the server.
$Server=$s.netname -replace '[\\\/\:\. ]', ' ' # remove characters that can cause problems
$instance = $s.instanceName -replace '[\\\/\:\. ]', ' ' # ditto
$DatabaseName =$database -replace '[\\\/\:\. ]', ' ' # ditto
$DirectoryToSaveTo=$DirectoryToSaveTo+$Server+'\'+$Instance+'\' # database scripts are local on client
if (!( Test-Path -path "$DirectoryToSaveTo" )) # create it if not existing
{ $progress ="attempting to create directory $DirectoryToSaveTo"
    Try { New-Item "$DirectoryToSaveTo" -type directory | out-null }
    Catch [system.exception]{
        Write-Error "error while $progress. $_"
        return
    }
}
}
<# now we will use the canteen system of SMO to specify what we want from the script. It is best to have a
list of the defaults to hand and just override the defaults where necessary, but there is a chance that a
later revision of SMO could change the defaults, so beware! #>
$CreationScriptOptions = new-object ("My.ScriptingOptions")

```

```

$CreationScriptOptions.Properties= $true # yes, we want these
$CreationScriptOptions.DRIAll= $true # and all the constraints
$CreationScriptOptions.Indexes= $true # Yup, these would be nice
$CreationScriptOptions.Triggers= $true # This should be included when scripting a database
$CreationScriptOptions.ScriptBatchTerminator = $true # this only goes to the file
$CreationScriptOptions.FileName = "$($DirectoryToSaveTo)$($DatabaseName)_Build.sql";
# we have to write to a file to get the GOs
$CreationScriptOptions.IncludeHeaders = $true; # of course
$CreationScriptOptions.ToFileOnly = $true # no need of string output as well
$CreationScriptOptions.IncludeIfExists = $true # not necessary but it means the script can be more
versatile
$transfer = new-object ("My.Transfer") $s.Databases[$Database]
$transfer.options=$CreationScriptOptions # tell the transfer object of our preferences
$scripter = new-object ("My.Scripter") $s # script out the database creation
$scripter.options=$CreationScriptOptions # with the same options
$scripter.Script($s.Databases[$Database]) # do it
# add the database object build script
$transfer.options.AppendToFile=$true
$transfer.options.ScriptDrops=$true
$transfer.EnumScriptTransfer()
$transfer.options.ScriptDrops=$false
$transfer.EnumScriptTransfer()
"All written to $($DirectoryToSaveTo)$($DatabaseName)_Build.sql"

```

This isn't entirely what we want for other purposes, of course. What about when you want to create a database without indexes, constraints or triggers, import the data in BCP fast-mode and then add the indexes, constraints and triggers? Yes, you can squirrel away far more test-runs this way, and load them rapidly, but in order to do it, you need a build script without them first, and a second build script with them only. With the first ones, you can have a 'knock-down' kill script that deletes everything from the database before you start, but you definitely don't want it for the second script. You'll soon be eyeing up all those scripting options, though, believe me. I'll be covering a lot about this in future articles.

## Automated scripting of objects

A second task is to save each object to a separate file. You'll need to do this to get your local (unshared) database into source control if you're not using SQL Source Control. The simplest way of doing this, if you are lucky enough to have SQL Compare, is

```
sqlcompare.exe /s1:MyServer /db1:MyDatabase /mkscr:MyNewDirectory /q
```

You can do it in Powershell, and you will, again, have more knobs you can twiddle to get the individual scripts exactly how you like them.

Here is a simplified script that shows you one of the several methods of doing this. Like SQL Compare, it saves each object type into its own subdirectory.

```

$ServerName='MyServer'# the server it is on
$Database='MyDatabase' # the name of the database you want to script as objects
$DirectoryToSaveTo='E:\MyScriptsDirectory' # the directory where you want to store them
# Load SMO assembly, and if we're running SQL 2008 DLLs load the SMOExtended and SQLWMIManagement Libraries
$v = [System.Reflection.Assembly]::LoadWithPartialName( 'Microsoft.SqlServer.SMO')
if (((($v.FullName.Split(',') [1].Split('=')[1].Split('.')[0] -ne '9') {
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SMOExtended') | out-null
}
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SmoEnum') | out-null
set-psdebug -strict # catch a few extra bugs
$ErrorActionPreference = "stop"
$My='Microsoft.SqlServer.Management.Smo'
$srv = new-object ("My.Server") $ServerName # attach to the server
if ($srv.ServerType-eq $null) # if it managed to find a server
{
Write-Error "Sorry, but I couldn't find Server '$ServerName' "
return
}
$scripter = new-object ("My.Scripter") $srv # create the scripter
$scripter.Options.ToFileOnly = $true
# we now get all the object types except extended stored procedures
# first we get the bitmap of all the object types we want
$all =[long] [Microsoft.SqlServer.Management.Smo.DatabaseObjectTypes]::all `
-bxor [Microsoft.SqlServer.Management.Smo.DatabaseObjectTypes]::ExtendedStoredProcedure
# and we store them in a datatable

```

```

$d = new-object System.Data.Datatable
# get everything except the servicebroker object, the information schema and system views
$d=$srv.databases[$Database].EnumObjects([long]0x1FFFFFFF -band $all) | `
    Where-Object {$_.Schema -ne 'sys'-and $_.Schema -ne "information_schema" -and $_.DatabaseObjectTypes -ne
'ServiceBroker'}
# and write out each scriptable object as a file in the directory you specify
$d| FOREACH-OBJECT { # for every object we have in the datatable.
    $SavePath="$($DirectoryToSaveTo)\${$_DatabaseObjectTypes}"
    # create the directory if necessary (SMO doesn't).
    if (!(Test-Path -path $SavePath )) # create it if not existing
        {Try { New-Item $SavePath -type directory | out-null }
        Catch [system.exception]{
            Write-Error "error while creating '$SavePath' $_"
            return
        }
    }
    # tell the scripter object where to write it
    $scripter.Options.Filename = "$SavePath\${$_name -replace '[\\\/\:\.}','-').sql";
    # Create a single element URN array
    $UrnCollection = new-object ('Microsoft.SqlServer.Management.Smo.urnCollection')
    $URNCollection.add($_.urn)
    # and write out the object to the specified file
    $scripter.script($URNCollection)
}
"Oh wide one, All is written out!"

```

This time, we get SMO's EnumObjects method for the current database object, then get the scripter object, via the Script method, to script out each object individually and save it to a separate file. Each filename is generated from the name of the object, and the directory name is generated from its object type. You can, of course, be very selective about what you script out and you'll see that scripting out a single object type, such as a table, is very simple.

In this following script, we'll save just the tables, scripting different schemas into different directories and adding the DRI, indexes, extended properties and triggers to each table. It would be equally simple to script whatever types of objects you want just by 'oring' the DatabaseObjectTypes to taste. ( in the previous script, I specified 'all but...')

```

$ServerName='MyServer'# the server it is on
$Database='MyDatabase' # the name of the database you want to script as objects
$DirectoryToSaveTo='E:\MyScriptsDirectory' # the directory where you want to store them
# Load SMO assembly, and if we're running SQL 2008 DLLs Load the SMOExtended and SQLWMIManagement Libraries
$v = [System.Reflection.Assembly]::LoadWithPartialName( 'Microsoft.SqlServer.SMO')
if (((($v.FullName.Split(',') [1].Split('=') [1].Split('.')[0] -ne '9') {
    [System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SMOExtended') | out-null
}
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SmoEnum') | out-null
set-psdebug -strict # catch a few extra bugs
$errorActionPreference = "stop"
$My='Microsoft.SqlServer.Management.Smo'
$srv = new-object ("$My.Server") $ServerName # attach to the server
if ($srv.ServerType-eq $null) # if it managed to find a server
{
    Write-Error "Sorry, but I couldn't find Server '$ServerName' "
    return
}
}
$scripter = new-object ("$My.Scripter") $srv # create the scripter
$scripter.Options.ToFileOnly = $true
$scripter.Options.ExtendedProperties= $true # yes, we want these
$scripter.Options.DRIAll= $true # and all the constraints
$scripter.Options.Indexes= $true # Yup, these would be nice
$scripter.Options.Triggers= $true # This should be includede
# first we get the bitmap of all the object types we want
$objectsToDo =[long] [Microsoft.SqlServer.Management.Smo.DatabaseObjectTypes]::Table
# and we store them in a datatable
$d = new-object System.Data.Datatable
# get just the tables
$d=$srv.databases[$Database].EnumObjects($objectsToDo)
# and write out each scriptable object as a file in the directory you specify
$d| FOREACH-OBJECT { # for every object we have in the datatable.
    $SavePath="$($DirectoryToSaveTo)\${$_DatabaseObjectTypes}\${$_.Schema}"

```



```

# create the directory if necessary (SMO doesn't).
if (!(Test-Path -path $SavePath )) # create it if not existing
    {Try { New-Item $SavePath -type directory | out-null }
    Catch [system.exception]{
        Write-Error "error while creating '$SavePath' $_"
        return
    }
}
# tell the scripter object where to write it
$scripter.Options.FileName = "$SavePath\${$_name -replace '[\\\/\:\.}','-').sql";
# Create a single element URN array
$UrnCollection = new-object ("My.urnCollection")
$URNCollection.add($_.urn)
# and write out the object to the specified file
$scripter.script($URNCollection)
}
"All is written out, wondrous human"

```

## Automated scripting of static data

The last task we'll set ourselves is to script out static data. This will really just include all those small tables without which your database simply won't work. Now, there is no way that anyone but you will know which tables this includes, but don't 'go ape' with this script, since it is scripting INSERT statements and that sort of thing doesn't scale effectively for big tables. No, sir: you'll want native-mode, fast-mode BCP for that.

This time, I've used a slightly different approach, in that I've actually constructed the URNs from the (maybe qualified) names of the table; this means the schema too if you specify it, and also the database if you want that too. You just specify what tables you want to script and we just go and do it. With SMO there are always several ways of getting to your destination.

```

# set "Option Explicit" to catch subtle errors
set-psdebug -strict
$DirectoryToSaveTo='MyDirectory'; # Local directory to save build-scripts to
$servername='MyServer'; # server name and instance
$Database='AdventureWorks'; # the database to copy from (Adventureworks here)
$Filename='MyFileName';
$TableList='product, AdventureWorksDW.dbo.DimCustomer, HumanResources.Department, person.countryRegion';
# a list of tables with possible schema or database qualifications
# Adventureworks used for this example
$ErrorActionPreference = "stop" # you can opt to stagger on, bleeding, if an error occurs
# Load SMO assembly, and if we're running SQL 2008 DLLs Load the SMOExtended and SQLWMIManagement Libraries
$v = [System.Reflection.Assembly]::LoadWithPartialName( 'Microsoft.SqlServer.SMO')
if (((($v.FullName.Split(', '))[1].Split('= '))[1].Split('.')[0] -ne '9') {
    [System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SMOExtended') | out-null
}
# Handle any errors that occur
Trap {
    # Handle the error
    $err = $_.Exception
    write-host $err.Message
    while( $err.InnerException ) {
        $err = $err.InnerException
        write-host $err.Message
    };
    # End the script.
    break
}
# Connect to the specified instance
$s = new-object ('Microsoft.SqlServer.Management.Smo.Server') $ServerName
# Create the Database root directory if it doesn't exist
$homedir = "$DirectoryToSaveTo\$Database\"
if (!(Test-Path -path $homedir))
    {Try { New-Item $homedir -type directory | out-null }
    Catch [system.exception]{
        Write-Error "error while creating '$homedir' $_"
        return
    }
}
$scripter = new-object ('Microsoft.SqlServer.Management.Smo.Scripter') $s
$scripter.Options.ScriptSchema = $False; #no we're not scripting the schema

```

```

$scripter.Options.ScriptData = $true; #but we're scripting the data
$scripter.Options.NoCommandTerminator = $true;
$scripter.Options.FileName = $homedir+$Filename #writing out the data to file
$scripter.Options.ToFileOnly = $true #who wants it on the screen?
$ServerUrn=$s.Urn #we need this to construct our URNs.
$UrnsToScript = New-Object Microsoft.SqlServer.Management.Smo.UrnCollection
#so we just construct the URNs of the objects we want to script
$Table=@()
foreach ($tablepath in $TableList -split ',')
{
    $Tuple = "" | Select Database, Schema, Table
    $TableName=$tablepath.Trim() -split '.',0,'SimpleMatch'
    switch ($TableName.count)
    {
        1 { $Tuple.database=$database; $Tuple.Schema='dbo'; $Tuple.Table=$tablename[0]; break}
        2 { $Tuple.database=$database; $Tuple.Schema=$tablename[0]; $Tuple.Table=$tablename[1]; break}
        3 { $Tuple.database=$tablename[0]; $Tuple.Schema=$tablename[1]; $Tuple.Table=$tablename[2]; break}
        default {throw 'too many dots in the tablename'}
    }
    $Table += $Tuple
}
foreach ($tuple in $Table)
{
    $Urn="$ServerUrn/Database[@Name=' $($tuple.database)']/Table[@Name=' $($tuple.table)' and
@Schema=' $($tuple.schema)']";
    $urn
    $UrnsToScript.Add($Urn)
}
#and script them
$scripter.EnumScript($UrnsToScript) #Simple eh?
"Saved to $homedir"+$Filename+', wondrous carbon-based life form!'
"done!"

```

Now we're making progress, and I'm hoping that, in these simplified scripts, I've given you some useful clues as to how to generate particular types of build scripts. I'll let you turn them into robust command-line tools with parameters and all the error handling, logging and other bits and pieces you'll want. In the next article, I'll explain how you can extend this functionality by actually executing SQL batches via SMO. This will allow you to automate migration scripts and do integration testing and all manner of other things all from the one script.

# Hadoop, NOSQL, and the Relational Model

Published Friday, February 10, 2012 1:28 AM

*(Guest Editorial for the IT Pro/SysAdmin Newsletter)*

Whereas Relational Databases fit the world of commerce like a glove, it is useless to pretend that they are a perfect fit for all human endeavours. Although, with SQL Server, we've made great strides with indexing text, in processing spatial data and processing markup, there is still a problem in dealing efficiently with large volumes of ephemeral semi-structured data.

Key-value stores such as Cassandra, Project Voldemort, and Riak are of great value for ephemeral data, and seem of equal value as a data-feed that provides aggregations to an RDBMS. However, the Document databases such as MongoDB and CouchDB are ideal for semi-structured data for which no fixed schema exists; analytics and logging are obvious examples.

NoSQL products, such as MongoDB, tackle the semi-structured data problem with panache. MongoDB is designed with a simple document-oriented data model that scales horizontally across multiple servers. It doesn't impose a schema, and relies on the application to enforce the data structure. This is another take on the old "EAV" problem (where you don't know in advance all the attributes of a particular entity) It uses a clever replica set design that allows automatic failover, and uses journaling for data durability. It allows indexing and ad-hoc querying.

However, for SQL Server users, the obvious choice for handling semi-structured data is Apache Hadoop. There will soon be an ODBC Driver for Apache Hive .and an Add-in for Excel. Additionally, there are now two Hadoop-based connectors for SQL Server; the Apache Hadoop connector for SQL Server 2008 R2, and the SQL Server Parallel Data Warehouse (PDW) connector. We can connect to Hadoop process the semi-structured data and then store it in SQL Server.

For one steeped in the culture of Relational SQL Databases, I might be expected to throw up my hands in the air in a gesture of contempt for a technology that was, judging by the overblown journalism on the subject, about to make my own profession as archaic as the Sagggar makers bottom knocker (a potter's assistant who helped the sagggar maker to make the bottom of the sagggar by placing clay in a metal hoop and bashing it). However, on the contrary, I find that I'm delighted with the advances made by the NoSQL databases in the past few years. Having the flow of ideas from the NoSQL providers will knock any trace of complacency out of the providers of Relational Databases and inspire them into back-fitting some features, such as horizontal scaling, with sharding and automatic failover into SQL-based RDBMSs. It will do the breed a power of good to benefit from all this lateral thinking.

by [Phil Factor](#)

# A Testing Perspective of Controllers and Orchestrators

14 February 2012

by Dino Esposito

The neat separation between processing and rendering in ASP.NET MVC guarantees you an application design that is inherently testable. It doesn't guarantee that your application will be well-designed and quick to test. For that, attention to use-cases and the structure of your code is essential.

This article focuses on the testability of ASP.NET MVC controllers and suggests that, if you keep your controllers super-thin and move the logic to services and model, then you have no need to unit-test controllers and probably don't need to mock the HTTP context that much. The article is a natural follow-up of a previous article that appeared on Simple-Talk... [Never Mind the Controller, Here is the Orchestrator](#).

## From RAD to Unit Testing

ASP.NET MVC is inherently testable. Testability, the degree to which software inherently supports testing, has been recognized as a fundamental attribute of software since the first draft of the international standard ISO/IEC 9126 paper about software architecture. The first draft of this paper dates back to 1991. However, Design for testability didn't seem to concern the average .NET developer much until 2004. Whatever the reason for its slow take-up, the success of .NET as a platform brought many companies to build more and more line-of-business applications, thereby dumping an incredible amount of complexity and business rules on development teams. Developers had to hurriedly change their approach to software development: Development needed to be rapid, but also reliable and extensible. It was becoming increasingly important to be able to design software in such a way to make it easy to test, and in particular to test *automatically*. Automated tests can give you a mechanical way to check edge cases and figure out quickly and reliably whether changes have broken existing features.

## Testing Applied to Controllers

When the long-awaited ASP.NET MVC was introduced, the framework made it possible for website developers to practice unit testing. A lot of tutorials have been posted to show you how to write unit tests around controllers. These tutorials assume that the controller is where you serve the request and coordinate access to backend layers such as the data access layer. If the controller is the nerve center of your ASP.NET MVC application then, once you can artificially create a HTTP context to simulate requests and responses to mock up the HTTP context, then you're pretty much done. Because ASP.NET MVC provides facilities to mock up the HTTP context, you are already provided with a clear and clean path ahead: You just need to write a bunch of unit tests for any of your controller classes and you'll be fine. Is this really true?

## Should You Test the Controller?

Testing is much easier if you can rely on clean code that is well separated in layers. Is the controller one of these layers? That question may surprise you if you, as an MVC developer, assume that the controller is the centralized machinery that governs the activity of an ASP.NET MVC site. The controller isn't really one of the architectural layers of an ASP.NET MVC site: more accurately, the controller is an essential element that's hard-coded in the framework. The controller is part of the infrastructure and therefore not a part of your code. The controller merely serves the purpose of receiving requests and dispatching them to other application specific services. [in a previous article of mine for Simple-Talk](#), I named these services as *orchestrators*. An orchestrator is a helper component that coordinates any activity related to a specific request. Ideally, an orchestrator is invoked by a controller but operates independently. This means that it receives from the controller any data that it needs to work on and returns to it any calculated data.

in this model, the controller is merely a pass-through layer with no significant logic that really needs testing. The real business is taking place in the orchestrator and so it is the orchestrator component where you will want to focus your testing efforts.

in general, you should focus more on the structure of your code and apply unit-testing practices where it is most beneficial. The code coverage, the percentage of code covered by unit tests, is a reliable indicator of neither code-quality nor the bug-count. If you use orchestrators, the controller is so thin that it needs almost no testing. It makes sense to take as much code as possible out of the controller classes by using orchestrators, because it helps you to focus on use-cases in sufficient detail to give these orchestrators a programming interface. This is a great stimulus to write better code because you are forced to plan the code in terms of use-cases and design issues. By using orchestrators you reduce significantly the need to mock the HTTP context. If, for example, some session state must be consumed by the orchestrator, then the controller will access it, extract any data and pass it to the orchestrator. in most cases, you can focus on testing the orchestrator without being overly concerned with the HTTP context.

## Testing Orchestrators

ASP.NET MVC does its best to support testing but it knows nothing about your application domain and the design you've come up with. ASP.NET MVC doesn't write tests for you either. You should aim at writing tests that are relevant rather than aiming at getting a high score in code coverage. What's the value in testing code like this?



```
[TestMethod]
public void TestIfSomeMethodWorks()
{
    var controller = new MyController();
    var result = controller.DoSomething();
    Assert.AreEqual(someExpectedResult, result);
}
```

The value of such a test is strictly related to the implementation of the method **DoSomething**. The method could be of any size; it might perhaps be executing a number of activities internally or might merely delegate the execution of the workflow to other layers. As I consider the controller to be part of the ASP.NET MVC infrastructure, I recommend the second approach. When the method **DoSomething** is just responsible for massaging some URL parameters and possibly some parts of the HTTP context into input values for orchestrators the value of testing **DoSomething** is much smaller than testing the orchestrator itself.

```
{
    private readonly IMyOrchestrator _orchestrator;
    public MyController(IMyOrchestrator orchestrator)
    {
        _orchestrator = orchestrator;
    }

    public ActionResult DoSomething( /* Parameters via model binding */ ...)
    {
        // Massage data from HTTP context into more abstract data structures
        var someSessionData = HttpContext.Session["Data"];

        // trigger the orchestrator
        var model = _orchestrator.PerformActionAndGetviewModel(someSessionData, ...);
        return view(model);
    }
}
```

The orchestrator is a class that receives anything it needs to work from the outside, typically via dependency injection (DI), either coded manually or via an IoC container.

```
public class MyOrchestrator
{
    public MyOrchestrator( /* List of dependencies */ ...)
    {
        ...
    }

    public SomeviewModel DoSomething( /* Data obtained from URL and HTTP context */ ...)
    {
        // Implement the workflow required BY the requested user action
        ...

        // use mockable dependencies here TO test in isolation
        ...
    }
}
```

The orchestrator class has no dependencies on the HTTP context. instead, it usually has a few dependencies on such services as repositories and domain services. These dependencies are easily managed via DI and can be easily mocked when it comes to testing. in other words, testing is simplified and there is complete decoupling of framework and logic. As a developer, you focus on UI triggers and those tasks that need to be orchestrated in response. You simply use the controller as the built-in mechanism that matches UI requests to tasks. To put it another way, the controller is not part of your code.

## When the Controller Orchestrates Itself...

Let's review a scenario in which the controller does it all. You seem to have a simpler project structure and coordinate any actions related to the requested task from within the controller. Your **DoSomething** method may look like below:

```
public class MyController
{
```

```
public ActionResult DoSomething( /* Parameters via model binding */ ...)
{
    // Massage data from HTTP context into more abstract data structures
    var someSessionData = HttpContext.Session["Data"];

    // Step 1
    // use some ICustomerRepository dependency

    // Step N
    // use some other dependency

    // Build a view model
    // This may be a long block OF code (30+ lines)
    return view(model);
}
```

The controller is instantiated by the registered controller factory. The default factory can only create controller instances by using their default constructor. To inject dependencies, therefore, you need to replace the factory. This is extra work.

in ASP.NET MVC 3 you can, perhaps, use dependency resolvers to save yourself the task of creating a factory. in terms of patterns, this means using a Service Locator based on IoC containers in order to inject dependencies. It works, and it seems elegant. I wouldn't say this is simple though. It rather seems to me quite convoluted. With dependency resolvers, you're writing less code yourself but you're forcing the framework to do a lot more work.

Finally, by putting this logic in the controller, you are distracted from use-cases and overall design. Writing logic in the controller is exactly the same as writing the logic in `Button1_Click`. This is what a controller method actually is: a more elegant way of rendering a button click handler. Clean design, and with it effective and really quick testing, is still far away.

## Final Thoughts

Personally, I have grown to love ASP.NET MVC. Mine was certainly not love at first sight. Love grew from appreciating the benefits of the neat separation between processing and rendering and the total control over the markup and responses. The care with which ASP.NET MVC was created was never extended to the tutorials. These, unfortunately, suggested to many developers that it was sufficient to merely use ASP.NET MVC in order to comply with best practices in architecture and design. It is certainly true, as the tutorials suggest, that testing is greatly simplified with ASP.NET MVC: Firstly, processing and rendering are neatly separated so you can intercept results being integrated in the view and write assertions on them. Secondly, some facilities exist to mock parts of a request that rely on the run time environment.

in ASP.NET MVC, smart tooling and framework magic make it possible for you to write code that can be unit-tested. Is it this also necessarily clean and well-designed code? I'd argue that ASP.NET MVC doesn't guarantee you a good design. If you miss this point, you are likely to produce controllers whose design is just an updated version of a `Button1_Click` handler, and which are, in consequence, difficult to test.

# Curing the Database-Application mismatch

Published Tuesday, February 14, 2012 7:52 AM

If an application requires access to a database, then you have to be able to deploy it so as to be version-compatible with the database, in phase. If you can deploy both together, then the application and database must normally be deployed at the same version in which they, together, passed integration and functional testing. When a single database supports more than one application, then the problem gets more interesting.

I'll need to be more precise here. It is actually the application-interface definition of the database that needs to be in a compatible "version". Most databases that get into production have no separate application-interface; in other words they are "close-coupled". For this vast majority, the whole database is the application-interface, and applications are free to wander through the bowels of the database scot-free. If you've spurned the perceived wisdom of application architects to have a defined application-interface within the database that is based on views and stored procedures, any version-mismatch will be as sensitive as a kitten. A team that creates an application that makes direct access to base tables in a database will have to put a lot of energy into keeping Database and Application in sync, to say nothing of having to tackle issues such as security and audit. It is not the obvious route to development nirvana.

I've been in countless tense meetings with application developers who initially bridle instinctively at the apparent restrictions of being "banned" from the base tables or routines of a database. There is no good technical reason for needing that sort of access that I've ever come across. Everything that the application wants can be delivered via a set of views and procedures, and with far less pain for all concerned: This is the application-interface. If more than zero developers are creating a database-driven application, then the project will benefit from the loose-coupling that an application interface brings. What is important here is that the database development role is separated from the application development role, even if it is the same developer performing both roles.

The idea of an application-interface with a database is as old as I can remember. The big corporate or government databases generally supported several applications, and there was little option. When a new application wanted access to an existing corporate database, the developers, and myself as technical architect, would have to meet with hatchet-faced DBAs and production staff to work out an interface. Sure, they would talk up the effort involved for budgetary reasons, but it was routine work, because it decoupled the database from its supporting applications. We'd be given our own stored procedures. One of them, I still remember, had ninety-two parameters. All database access was encapsulated in one application-module.

If you have a stable defined application-interface with the database (Yes, one for each application usually) you need to keep the external definitions of the components of this interface in version control, linked with the application source, and carefully track and negotiate any changes between database developers and application developers. Essentially, the application development team owns the interface definition, and the onus is on the Database developers to implement it and maintain it, in conformance. Internally, the database can then make all sorts of changes and refactoring, as long as source control is maintained. If the application interface passes all the comprehensive integration and functional tests for the particular version they were designed for, nothing is broken. Your performance-testing can "hang" on the same interface, since databases are judged on the performance of the application, not an "internal" database process. The database developers have responsibility for maintaining the application-interface, but not its definition, as they refactor the database. This is easily tested on a daily basis since the tests are normally automated. In this setting, the deployment can proceed if the more stable application-interface, rather than the continuously-changing database, passes all tests for the version of the application.

Normally, if all goes well, a database with a well-designed application interface can evolve gracefully without changing the external appearance of the interface, and this is confirmed by integration tests that check the interface, and which hopefully don't need to be altered at all often. If the application is rapidly changing its "domain model" in the light of an increased understanding of the application domain, then it can change the interface definitions and the database developers need only implement the interface rather than refactor the underlying database. The test team will also have to redo the functional and integration tests which are, of course "written to" the definition. The Database developers will find it easier if these tests are done before their re-wiring job to implement the new interface.

If, at the other extreme, an application receives no further development work but survives unchanged, the database can continue to change and develop to keep pace with the requirements of the other applications it supports, and needs only to take care that the application interface is never broken. Testing is easy since your automated scripts to test the interface do not need to change.

The database developers will, of course, maintain their own source control for the database, and will be likely to maintain versions for all major releases. However, this will not need to be shared with the applications that the database servers. On the other hand, the definition of the application interfaces should be within the application source. Changes in it have to be subject to change-control procedures, as they will require a chain of tests.

Once you allow, instead of an application-interface, an intimate relationship between application and database, we are in the realms of impedance mismatch, over and above the obvious security problems. Part of this impedance problem is a difference in development practices. Whereas the application has to be regularly built and integrated, this isn't necessarily the case with the database. An RDBMS is inherently multi-user and self-integrating. If the developers work together on the database, then a subsequent integration of the database on a staging server doesn't often bring nasty surprises. A separate database-integration process is only needed if the database is deliberately built in a way that mimics the application development process, but which hampers the normal database-development techniques. This process is like demanding a official walking with a red flag in front of a motor car. In order to closely coordinate databases with applications, entire databases have to be "versioned", so that an application version can be matched with a database version to produce a working build without errors. There is no

natural process to “version”™ databases. Each development project will have to define a system for maintaining the version level.

A curious paradox occurs in development when there is no formal application-interface. When the strains and cracks happen, the extra meetings, bureaucracy, and activity required to maintain accurate deployments looks to IT management like work. They see activity, and it looks good. Work means progress. Management then smile on the design choices made. In IT, good design work doesn’t necessarily look good, and vice versa.

by [Phil Factor](#)



# Seth Godin: Big in the IT Business

02 February 2012

by Richard Morris

Seth Godin has transformed our understanding of marketing in IT. He invented the concept of 'permission marketing', sees the end of the "TV-Industrial complex" and the techniques of 'interruption-marketing': Instead he sees a sunny future, one where the consumer has the power to drive sales on merit.

Even a decade ago, during the long forgotten days of high-waisted jeans, Seth Godin was lionised and spoken of as a minor deity among online marketing folk.

It was some years before advertising wizards would go virus mad, unleashing an epidemic of outright mediocrity on an undeserving public after misunderstanding his book 'Unleashing the Ideavirus' – it quickly became the most popular e-book ever published. The irony was lost on some.



Technology has played a big part in Seth's life. He graduated from Tufts University near Boston with a degree in computer science and philosophy, then went on to lead a team at Spinnaker Software which created computer games for the 8-bit Commodore 64.

The games, Godin says, were so large that they needed four floppy disks apiece and the projects so complex that smoke more than once began coming out of the drives.

He remembers that in the 1980s product success with software was measured by an absence of fires and the ability to avoid a nervous breakdown.

Now the author of thirteen books, he has splashed through the post-industrial revolution, the way ideas spread marketing, quitting, leadership and most of all, changing everything. Each book has been an international bestseller, translated into more than 20 languages.

He's responsible for debunking traditional ways of marketing, an industry which has had so often tendrilled itself with the bleeding obvious and given the English language some the most impenetrable meanings in the marketer's handbook including permission marketing, ideaviruses, purple cow, the dip and sneezers.

For some his writing draws parallels with the nineteenth century – he possesses a Victorian work ethic, he is a 21st century Samuel Smiles whose own book *Self Help*, was published to advocate the benefits of "painstaking labour" and unremitting study". Seth Godin could well have written that 'every human being has a great mission to perform, noble faculties to cultivate, a vast destiny to accomplish.'

As an entrepreneur Seth has had his share of failures and successes. His first business Yoyodyne pioneered the use of ethical direct mail online.

When Yoyodyne was bought by Yahoo! in 1998, he became VP of Direct Marketing for a year before leaving to become a full time speaker, writer and blogger.

He was one of the first to realise that the media world - the big, official one where reporters wear sensible shoes - was becoming less of a distinct entity and that this generation; more so than any before it, thrives on communication and the manic urgency of change and reform.

A regular at Business of Software conferences, his latest company Squidoo.com, is ranked among the top 125 sites in the US (by traffic) by Quantcast. It allows anyone to build a page about any subject.

He holds an MBA from Stanford (which he completed while holding down his job at Spinnaker Software) and was called "the Ultimate Entrepreneur for the Information Age" by Business Week. Seth lives in Hastings-on-Hudson, New York with his wife Helene and their two sons.

**RM:** Seth, your first real job was leading a team at Spinnaker Software which created computer games for the Commodore 64, among other things. How and why did you get the job?

**SG:** I got the job the way most people do... sort of as an accident. I had been offered a job as assistant to the President of Activision, at the time the fastest growing company in the history of the world. As a summer job from Stanford Business School, what could be better? My goal was fast, not software. While I had studied computer science (ht to George Meyfarth, wherever he is) I wasn't very good at it.

Well, family intervened and I ended up in Boston with weeks to go before the summer and no job. Lucky for me, when I called the company (Spinnaker Software) late in the evening, the chairman of the company picked up the main number. He invited me in, hired me for the summer and then... on my first day, realized he hadn't told anyone I was coming. Awkward.

It went uphill from there.

**RM:** What were the things you had to learn about industrial software and programming?

**SG:** It's like architecture and contracting mixed together, except you can't see the progress easily and more hands do not make things go faster. A lot of people think they know what they're doing, few do. The biggest insight: devote all your resources to helping the people who do.

**RM:** I was reading *Poke the Box* which explains why it is important to keep poking away and inventing new ways to work, coming up with new ideas and new products but most importantly to get the idea or product out there.

Talking specifically about technology the counter argument to more products is that the invention of new technology seems to be totally random. Some things get better, some worse. It's like evolution but isn't goal-directed.

I suppose the most obvious example is something really simple like the USB. The universal protocols it dictates require a huge amount of software to support. To develop such software requires reading thousands of pages of specs. Which are neither complete nor accurate. So then someone develops an interface chip that encapsulates the complexity, and then you must learn to use that chip, which is at least as complex.

Why is it so tempting to solve problems that we don't really have?

**SG:** What do you mean by "really"? Brown rice, a leakproof roof and some shoes and you have few problems left, right? All that other stuff, that detritus, is merely amusement. Our economy is built on want, not need.

**RM:** There has a lot of focus on Code Year, which is to try and get more people to realize the importance of programming. A lot of people have signed up including the New York City Mayor Michael Bloomberg.

There's an argument which starts that programming teaches a way of thinking that's important, it gives you order and structure in life.

Opponents to this, say programmers are misunderstanding the world in exactly the same way everybody else does and programming doesn't make you intellectually superior. What camp are you in?

**SG:** "Intellectually superior?" There's a loaded term. I think it's simple: any time you learn a way of thinking that is useful for a productive portion of the population, you've done something brilliant. There's no one way, no right way, but this is sure a good way.

I taught intro computer science in college, the youngest person in the engineering school to do so. And I never got much better at it. But there's no doubt that I learned a huge amount about how to think.

Now we need to get programmers to start writing and painting and reading poetry. Then we'll be even.

**RM:** Do you feel that academic computer science and industrial programming meet in the right place? Are there areas where academics are out in front of industry? Where industry is ignoring good stuff about how we ought to build software.

**SG:** My limited experience with the state of the art of computer science feels a lot like the way business is taught in business school—don't go there if you actually want to learn useful skills for everyday life.

There's no doubt that some practical breakthroughs will occur, but my guess is that 100x as many breakthroughs come from programmers doing actual work.

For the real breakthroughs we should look to computer science for are ones that change the questions we're asking, not ones that answer them.

When I was in high school, no one told me that people had written down algorithms. So I "invented" the bubble sort, from nothing. I still remember how it felt when I discovered that it worked...

**RM:** Why do you think it is that people these days measure a computer scientists by standards of dollars or by applications beauty rather than measure the contributions to knowledge, even though contributions to knowledge are the necessary ingredient to make previously unthinkable applications possible?

**SG:** Partly because the computer science guys asked to be measured that way. In many cases, they promise real world innovation right around the corner. But that's an engineering concern, not an academic one.

**RM:** Do you still think that computing needs a 'grand challenge' to inspire it much in the same way that the lunar challenge in the 1960s sparked a decade of collaborative innovation and development in engineering and space technology?

**SG:** I actually don't think the space challenge was the only cause of the explosion of innovation that occurred. A big part of it was that science and engineering were ready, that computers were coming online and that DARPA was very active.

These are the good old days. Right now. Ultra connected, highly leveraged.

**RM:** You're rightly known as a great marketer and as an author. Do you try to write books that will make a fortune for people or try to imagine which books are needed most, or which books would be the most fun to read?

**SG:** None of these. I write to satisfy the voice in my head, to spread ideas I care about and to help people see an outcome or possibility they might not have considered before.

**RM:** Has any philosopher influenced your work?

**SG:** Many of them, some obscure and some not taught in philosophy class. Doug Hofstadter, Dan Dennett, Richard Dawkins, Steve Pressfield, Pema Chodron, and my buddy Zig Ziglar. And maybe Dr. Who and Mr. Spock...

**RM:** Would you still publish books whether or not it had any huge commercial value?

**SG:** Of course! They actually don't have huge commercial value. I've created more shareholder value with my two internet companies than I will with the next 100 books I write. Authors and publishers get in trouble when they pretend that this is a business. It's not. For every JK Rowling, there are literally 100,000 failed writers. Is there any real industry where this is true?

**RM:** Do you have any long-range ambitions or regrets as a writer? Have you already written the greatest book of your life or is it still to come? Do you throw away a lot of ideas?

**SG:** Yes, maybe and yes.

**RM:** Which book do you think contains your best ideas?

**SG:** Survival is Not Enough combines Darwin with rapid loops of change in today's organizations. And Linchpin is the book that resonated with the most people, arguing that compliant cogs are facing a rough road ahead.

**RM:** You mentioned to me prior to this interview that Don Knuth was someone you admire. What do you most admire about him? Have you read The Art of Computer Programming?

Some people I've interviewed have read it from cover to cover; some have it on a shelf for reference. And some people just have it on a shelf.

**SG:** Academia often pretends it wants a singular voice to arise, a bold breakthrough and a gutsy artist to take a chance and lay it out for us. Too often, this just isn't true. Too often, it's about meetings and committees and fear, just like real life. I remember when I studied CS at Tufts and saw his work and said, "this guy isn't afraid."

**RM:** When you look back at your career on all the things you have done is there one time or a period that stands out among all the others?

**SG:** I am amazed at how formative the years were years ago. How lucky I was to escape the bad breaks I had. How persistent I had to be to get over the hump. And how much leverage there can be on the other side.

This is available to anyone who wants to sharpen a pencil and find an appropriately sized project. The magic of collaboration and worldwide instant publication is that yes, important ideas are going to spread, and I have no doubt your readers have more important ideas than I do.

# Scott Shaw: DBA of the Day

24 January 2012  
by Richard Morris

Scott Shaw was one of the finalists to the 2011 Exceptional DBA Award (XDBA). The award was founded in 2008 to recognize the essential but often overlooked contributions of DBAs, the unsung heroes of the IT community. In this interview, Scott describes the challenges of being a DBA in a busy Healthcare company, and his work for the DBA community

Scott Shaw's IT career began in the late 1990s, when he was given an opportunity to work for a small consulting firm. From there, he was swept up in a maelstrom of IT startups and, though it was here that he first came across SQL, it took another five years working variously as a server architect, network architect and Oracle DBA before he finally settled down as a full-time DBA.



A well-known member of the SQL Server community, Scott has spoken at various SQL events, including the first SQL Rally in Orlando. He is a regular blogger ([blogofshaw.blogspot.com](http://blogofshaw.blogspot.com)) and he occasionally teaches at Washington University on the Center for the Application of Information Technology (CAIT) program.

He now works as a full-time SQL Server team-lead for a healthcare company in Saint Louis.

Do you think the accidental route is still the most prevalent path to becoming a DBA?

**RM:** Though I don't have any research numbers to quantify my statement, I'd still say the accidental DBA is the most prevalent path to becoming one. I've worked with people who have had DBA work thrust on them and my experience has been that they may complain but they don't really mind. They know and understand the value of that experience.

As a consultant, I've worked with companies with accidental DBAs, and they are rarely willing to give over the DBA work even though it isn't their primary function. I've also seen a lot of people become DBAs through internal hiring. Anytime we have an opening in our company for a DBA it generates a lot of buzz. I like to think it's because I'm the team lead and people would love to work with me, but I have to be realistic and know that the title of DBA generates respect in the IT community and is also a good career move.

**RM:** Is it unusual to see people queuing up wanting to be a DBA?

**SS:** It's not unusual for someone to transfer to our DBA team from service desk or server operations, work a couple of years as a DBA and then move to a job with a 20% to 30% pay increase. When they leave they become a professional DBA and it's their career for life.

That being said, I think being a DBA is one of the best paying careers someone can have without requiring any formal training. I have yet to hear of a university or any technical school offering training specifically to become a DBA - obviously I'm excluding the wealth of training you can get through the SQL community, but even that training won't provide you with a degree. They may provide classes on data modelling, data design or database concepts, but there is no formal degree for being a DBA.

**RM:** Can you tell me a little about your job and how this differs from other DBA work that you know of? What database software do you use?

**SS:** My current job as a DBA doesn't differ much from my other DBA jobs. It still has the same requirements and some of the same challenges. What is different is the scope of the work and the type of databases we support. I'm in healthcare, so the industry exposes me to a lot of unique environments.

We have databases for hospital food distribution, databases for pre-natal care, databases for electronic records and databases for patient monitoring. In fact, we have over 300 unique applications running on over 400 SQL Server instances.

Management is the same for individual servers, but it needs to be spread out. Automation is always a challenge and we are constantly reviewing tools and ideas to help automate our tasks. We worked with PowerShell, Red Gate Toolbelt and Idera products.

We also use custom T-SQL scripts, SSIS, MDW and pretty much anything else we can get our hands on. I'm always trying to improve the environment to make it more manageable. It's a difficult challenge and it's true when they say that change is the only constant.

**RM:** What about future opportunities? What excites you? Are there aspects of technology you work with, or your job, that make you fed up?

**SS:** I'm excited about the future opportunities. Data is growing exponentially and with increased needs, so I think if you work with databases you have a good future career ahead of you. There are significant challenges. You have to enjoy learning. You can't rest as a DBA and maintain the status quo. You need to stay ahead of the curve and keep your skills modern.



One thing I find depressing is when I witness upper-management and executive level people who have very little technical skills or background and who view database administration and support as a commodity or necessary evil. I think this is true especially for SQL Server. I've had a number of management people outside of the database department make a sarcastic comment about how difficult it is for a SQL DBA to simply spend all day clicking "next...next...close".

They don't understand how costly it can be for an organization to not take care of their data, or make sure access to data is running well and is available. Some do get it but most don't. SQL Server has been relied on for years as the easy database to set up and deploy (particularly when compared to Oracle). As Microsoft is trying to move into the data warehouse market and as databases are getting bigger and bigger, I'm seeing many executives hesitant to regard SQL Server as a viable solution. As a SQL Server DBA I cringe when executives look at my role in the company as second-class and under-appreciated.

**RM:** Are there things in your career that you would have done differently?

**SS:** In all honesty, I don't know of any way I could have done things differently. I guess looking way back I was interested in literature early on and wanted to be a teacher, so I attained a Master's Degree in Literature.

I don't necessarily regret doing that because it helped shape who I am, but maybe if I could do it again I would have been more practical. It's ironic, because in the mid-1990s, right when Microsoft was taking off, I was living in Seattle, but as an artist wanna-be and not as a techie. I wish I had the foresight to have gotten involved in computer work back then - my older brother was into computers from the very beginning and still is, so I had plenty of access to them.

If I had started in technical work earlier then things would have been much different and I might have spent less time being hungry and poor. It wasn't until my 30's that I went back to school and received an MS in Information Systems Management. Trust me: it is a lot harder to get a degree when you're 35 with kids than when your 20 with no kids.

**RM:** What would you say are the main characteristics of an exceptional DBA? Why do you think these qualities are so important?

**SS:** I see three main qualities of an exceptional DBA. The first is a thorough understanding of the technology. You have to know your stuff, that's the foundation. Second, you have to be good with your customers. You have to be able to take the technical foundation and transfer it into good troubleshooting skills. Finally, you have to be involved in the community. You have to get out of your cubicle and give back.

An exceptional DBA is willing to put themselves out into the community to help others. They need to overcome any fears of criticism or public speaking and give back. For me, taking time voluntarily to give back to the DBA community really is the exceptional part.

**RM:** What achievements are you most proud of?

**SS:** Does living this long count as an achievement? I've had some close calls.

**RM:** Have you been a stunt double or something?

**SS:** I wish! If I was then I would have least been paid for my self-destructive behaviour. In reality I was just very careless growing up. I was a bit of free spirit and probably took the starving, binge drinking literary lifestyle a bit too seriously. At one point in my life I just assumed all my work would be published posthumously.

Actually, my number one achievement was making the career switch to IT. I cannot express enough what it has done for me in my life. It has allowed me to provide for my family way beyond anything else I could have done at the time, as my options were very limited.

When I look back on all the jobs I went through and the effort I went through to get a degree, I'm amazed I am where I am. I feel lucky and blessed. I've had a lot of people help me along the way and I'm grateful for them. It is a bit frightening that I may have another 30 years of this. Who knows what the industry will look like in the year 2040. I'm still proud of what I've accomplished so far.

**RM:** What do you see yourself doing in 30 years time?

**SS:** I've considered this a lot and don't have an answer. A pension is out of the question so there is little incentive to stay in one spot, doing the same thing for a long time. The question is whether I will want to stay in a large company or move to a smaller one, or whether I want to continue down a technical track or move into a more managerial role.

I think some of these questions will be answered for me, but for others I may have a choice. I enjoy getting involved in the SQL community but, to be honest, it can be very stressful to do community work while still maintaining a full-time job and raising a family. Still, the community work I'm doing now is a long-term plan and something I hope to do far into the future.

Technology will also affect my course. If you think of how much technology has changed in the last 30 years its daunting to think where we will be in the next 30 years. Adaptability is critical. In order to last another 30 years in IT you'll need to constantly adapt and always keep an eye out for new opportunities.

**RM:** Why do you think it's important to participate in the SQL Server community?

**SS:** Well, this question is asked all the time and the answer is always the same. I want to give back. But also, I think the more we talk and gather as DBAs the stronger our industry becomes. There is no DBA union and for me the community is the

closest we can get to organizing. I don't mean so that someday we can strike, but instead, we gather so we can make each other better. The sum is greater than the parts.

As I mentioned before, there is no DBA degree so the community represents that common knowledge bringing us together in a common trade. We learn best practices and skills based on others who have already been through the journey and it helps define what it is to be a DBA. No one else can define it for us.

Involving yourself in the community is a very personal decision. I guess you can say I've fulfilled most of Maslow's basic needs and it was time for me to self-actualize and work towards more abstract needs. Like I said previously, I've got 30 more years of this and I get bored easy, so what better thing to do then jump into the SQL community and prop myself up for ridicule?

**RM:** It's said that most DBAs live chaotic lives, in that they always seem to be on call. Is it the same for you and have ever considered a change of career?

**SS:** Believe it or not, my current position has a slower on call than any other job I've had. Don't get me wrong, I've paid my dues and this job still has some nasty situations. When things go down in a hospital it isn't good. Still, the chaos in my life is more with raising children than with being a DBA. I also have a wonderful team and we help each other out, so no one person is saddled with all the work. Our on call rotation lasts a week and there are 5, of us so I'm only on call every fifth week. That's not bad.

I wouldn't say I would change careers, I'd say more like I would have no career at all. I love my work but I love even more just doing what I want to do. But then I'm not independently wealthy, so I'll stick with being a DBA. With that being said, I do love being a DBA and it's a great career to have. I feel privileged to have been able to maintain it and succeed at it. My son is getting to the age where he starts thinking of what he wants to do when he grows up and he has said he wants to do what I do (I don't think he quite understands what I do – kind of like my manager). I think being a DBA is a career I'd recommend to him.

**RM:** How would you make money from your skills if you weren't in the job that you now?

**SS:** Good question. If the bottom completely fell out of IT tomorrow and all database work was suddenly automated and performed by glowing knowledge cubes, I would probably start a company providing 24/7 support for glowing knowledge cubes. On second thought, I would go into healthcare IT. I would go back to school to get a quick degree in hospital administration and then work technical support for hospital functions. I think it would be an exciting and challenging area to work in. It may not pay as well but it would certainly be a growth industry. Another option would be BI and data warehousing. Healthcare especially will be showing exponential growth in BI needs due to Meaningful Use requirements.

**RM:** Is there any moment or event, either in IT or computer science, you would like to have been at, and why?

**SS:** I would have liked to have been a secretary with stock options when Microsoft first started. I also would have liked to have been working at Facebook when it first started. Now that looked like a party. I would have loved to have been at the start of any of the major IT companies. Companies have lifecycles and it would have been great to be part of the beginnings of something like Google. It would have been great to be part of the vibrant, intellectual atmosphere, and knowing that you were part of something influential. You were working hard, being creative and making money but, most of all, having a fun time.

# ANTS Performance Profiler 7.0 has been released!

Published Thursday, February 16, 2012 2:58 PM

Please join me in welcoming ANTS Performance Profiler 7 to the world of .NET.

ANTS Performance Profiler is a .NET code profiling tool. It lets you **identify performance bottlenecks** within minutes and therefore enables you to optimize your application performance.

Version 7.0 includes integrated decompilation: when profiling methods and assemblies with no source code file, you can generate source code right from the profiler interface. You can then browse and navigate this automatically generated source as if it was your own. If you have an assembly's PDB file but no source, integrated decompilation even lets you view line-level timings for each method, pinpointing the exact cause of performance bottlenecks. Integrated decompilation is powered by [.NET Reflector](#), but you don't need Reflector installed to use the functionality. Watch this [video](#) to see it in action.

## Also new in ANTS Performance Profiler 7.0:

• Full support for SharePoint 2010 - No need to manually configure profiling for the latest version of SharePoint

• Full support for IIS Express

• Azure and Amazon EC2 support, enabling you to profile in the cloud

Please [click here](#), for more details about the ANTS Performance Profiler 7.0.

by [Michaela Murray](#)

# Geek of the Week: Don Syme

01 February 2012  
by Richard Morris

With the arrival of F# 3.0 Microsoft announced a wide range of improvements such as type providers that made F# a viable alternative to their other .NET languages as a general purpose workhorse. So what exactly are type providers, and why are they a killer reason for using F#? Why should we be considering F# for data-rich applications? To find out, we caught up with Don Syme, F#'s creator, to ask him about the latest developments in F# 3.0 and canvas his views on functional programming in general.

F# has always been a fascinating .NET language. You can easily mistake Visual Basic for C# nowadays, and even PowerShell's idiosyncrasies are getting smoothed out, but you can't mistake F# for any other leading .NET language.



Until its full integration into Visual Studio in late 2005, F# may have seemed to be a specialised language for those esoteric purposes where analytical programming is suitable. But, by combining F# Interactive with Visual Studio, F# users became able to develop fast, accurate code using Visual Studio's background type-checking and IntelliSense, while exploring a problem space using F# Interactive.

Now that F# 3.0 has introduced the idea of type providers using LINQ, it has grown into a strong candidate for a wide range of data applications. F# 3.0 aims to be able to integrate data from a variety of external information sources, such as web information, web services, data markets and enterprise data, in a way that is extensible and strongly-typed. It is then able to inter-operate with them using the power of Visual Studio.

So, what exactly are type providers, and why does Don Syme reckon they're so important that C# and VB will soon be using them? Why should we be considering F# for data-rich applications? To find out, we caught up with Don over a cup of coffee, to ask him about the latest developments in F#3.0 and canvas his views on functional programming in general.

**RM:** Don, a lot has been happening with F# since we last spoke two years ago. What are the latest big developments in F#?  
**DS:** The language is certainly developing fast, and we're very glad with the direction we have. First, the F# community and user base has been growing steadily, and we're seeing good adoption of F# 2.0 in the areas of analytical programming that we've been focusing on. We've also been seeing great action from the F# community in expanding out the capabilities of F# in server-side and client-side programming.

Technically, the major announcements we made at the BUILD conference were about F# 3.0, especially about a set of features called F# Information Rich Programming. This technology lets you program directly against rich spaces of data and services such as databases, web services, web data feeds and data brokers.

You can think of F# Information Rich Programming as a kind of LINQ-on-F#-steroids. As part of this work, we've integrated LINQ queries into the language, and we've added a new, unique feature called type providers.

Importantly, these F# 3.0 data-access features will also be accessible to C# and VB programmers simply by adding an F# project to their solution.

**RM:** Can you tell me what the aim of type providers is, what problem will this solve and how will it give tighter integration between the F# programming language and external data sources?

**DS:** Type Providers are about replacing our conventional notion of a "library" with a provider model. This allows a type provider to project an external information source into F# and makes it easier to access diverse sources of data. Type providers for several commonly used data sources are included in the F# library, including SQL databases (both LINQ-to-SQL and LINQ-to-Entities), OData feeds and WSDL services.

Importantly, Type Providers integrate very nicely with IntelliSense, auto-complete, scripting and explorative programming. That is, when you program with a database, web-service or other data source in F#, you're doing strongly-typed programming.

Type providers also scale to enormous external data services such as Azure DataMarket and web ontologies such as Freebase or Dbpedia. These data sources contain literally thousands of types and more are being added and organized every day.

**RM:** I remember you saying at PGD that this work is the most important you've ever done beyond the implementation of generics into .NET. The obvious question is, why is this important?

**DS:** For me, part of the role of F# is about proving that statically-typed languages can play fully in the modern world of connected programming, without losing the simplicity, elegance or tooling that come with strong types. Type providers are



an essential part of tackling this, because we can no longer ignore the information-richness of external data sources, and have to change language and compiler architecture to adapt.

**RM:** Do you think that the changes in F# have relevance for all typed languages over time? Is this development more Behavior-Driven Development (BDD) than Test Driven Development (TDD)?

**DS:** Yes, I do think this has long-term relevance. I'll go out on a limb and say that I expect most major, forward-looking typed languages to have type-provider-like architectures in the coming years.

That said, F# Information Rich Programming combines particularly well with the highly type-inferred nature of F#, with F# scripting and with the visual tooling of F# such as auto-complete.

**RM:** F# adds huge value to the .NET platform and enables the platform to reach out to some new classes of developers, and appeal to some domains where .NET is not heavily used. Is F# going to be central to Microsoft's enterprise strategy in the future?

**DS:** The modern enterprise has a range of software needs that don't fit the traditional "business apps" IT model of WinForms and WPF. For example, file processing, server-side programming, technical computing, data analysis, parallel programming and cloud programming all require very different skills to those associated with the OO languages of the 1990s.

In the F# team we call these problems "analytical programming", in other communities people might say "code behind", or "business logic" or the like. Analytical programmers are always on a search for tools which improve productivity, performance and robustness.

For these problem domains, F# is a central part of the tools we provide in Visual Studio to allow people to solve their real-world, practical problems.

**RM:** How will the advances in C++ affect the future of F#?

**DS:** It's great to see C++ developing as a language, and of course it is an essential tool for purposes like systems and games development. However, I see very few people moving back to C++ once they have embraced languages like F#, unless they need it for a specific purpose.

The special thing about F# is that we manage to get most of the benefits of dynamic language, like rapid development, data-richness and succinctness, while also still retaining the benefits of strongly-typed languages such as robustness, performance and visual tooling. Further, F# programming is renowned for having low bug rates. And all in the context of a highly interoperable platform.

**RM:** One of the key things about F#, of course, is that it spans the spectrum from interactive, explorative scripting to component and large-scale software development. Was this always a key part of the development of F#, or has it simply morphed into a language with these added to it?

**DS:** When I look back, I only think that F# really found its first cohesion as an overall concept in around 2006-2007. At this time, we started to describe how "it's the combination that counts" - the combination of succinctness, efficiency, libraries, strong-typing and so on. We didn't have all of these in place from day one, but when they came together, we knew we had a real contribution to make to programming languages.

**RM:** Functional programming is gaining in popularity and not just among the research community, but there are still people who see functional programming as being driven by ideas that, while excellent, can be very mathematical and divorced from day-to-day programming. Is that a fair characterisation?

**DS:** There are certainly people who think like that, and I think there always will be.

Equally, functional programming is now so much a part of C# and Javascript (e.g. JQuery) that there are many people doing at least some functional programming all day long, and often barely even realizing it.

That said, there are plenty of ideas floating around amongst both object-oriented and functional programming researchers which are way too "out-there" for immediate adoption by the industry. I don't mind if people take their time to adopt and learn these things - and I do think that functional programming needs to be used only when appropriate, and requires good training, knowledge and experience.

**RM:** The culture of the programming language community tends to be 'prove that your type system is sound and complete'. Do you think in practice functional programs make people more productive? For example, are you more productive writing a functional program or an object-oriented program to do the same thing?

**DS:** In F#, it is the synthesis of functional programming and OO programming that makes people more productive. Here, functional programming means type inference, mutable-state-minimization, closures and simple-orthogonal language features where possible. I am convinced these make people more productive. We're seeing many examples of that in practice.

For most tasks, I am less productive in a language that supports functional programming alone, just as I am less productive in a language that only supports OO programming.

**RM:** I remember Simon Peyton Jones once telling me that at Microsoft in Redmond, systematic attempts are made to watch

programmers who have been given a new API talk through what they are trying to do, with the people who designed the API sitting behind a glass screen watching them. The idea is that the API is changed if necessary. Have you used this particular method with F# and/or is programming language research weak on that score regardless of who designed the language?

**DS:** We've used this technique as part of the F# product team, and it's simple and great.

I don't think it is at all normal to use this technique in any programming language research. Some researchers do try to do some empirical studies of language features, but very few do in-situ informal usability studies. Traditionally, programming language researchers have largely ignored or avoided these "human" aspects of their work – mostly because it is extremely difficult to do convincing controlled experiments in this space. All users come with background and "baggage" which makes them see a new language or API in a particular way.

**RM:** What advice do you have for up-and-coming programmers?

**DS:** My advice would be to learn F#, Python, Prolog, Haskell, C# and Erlang!

# Using SQL Test Database Unit Testing with TeamCity Continuous Integration

02 February 2012  
by Dave Green

With database applications, the process of test and integration can be frustratingly slow because so much of it is based on manual processes. Everyone seems to agree that automation of the process provides the answer to accommodating shorter development cycles, but how, exactly? Dave Green describes a successful process that integrates third-party tools.

It is a tenet of extreme programming, and other development methodologies (e.g. Test Driven Development and Agile) that [Continuous Integration](#) (CI) processes should be used. Continuous Integration means that the development cycles are short, and each small set of code changes is tested with the existing code base, and released (to other developers) quickly. This includes unit testing on a near-continuous basis. Using CI means that development is being done against the latest code at an earlier stage. The idea is to save time overall by spotting and fixing problems early ("a stitch in time saves nine"). It moves the process of quality control to an earlier point in the development process, which allows higher quality code to be developed more quickly, as well as alleviating the difficulties of merging significantly different code branches.

Whilst CI has been used for a number of years within some software languages (e.g. C#, Java), the idea has been adopted less widely with database software, not least because of the difficulty of obtaining [unit testing](#) and source control tooling for SQL Server database objects.

The model I will be working towards in this article is to put database objects (tables, stored procedures, data, etc.) into source control, add unit tests to the database then get the CI engine to rebuild a copy of the database, and run tests on a periodic basis to detect any issues with the software.

I've been using Red Gate's [SQL Test](#) product which, at the time of writing, is in Preview, together with a [TeamCity](#) Continuous Integration development process. I use TeamCity because it is available under both a free and commercial license (lowering cost of trying it out, but giving the ability to pay for support once adopted), its compatibility with a number of version control systems and programming languages, and the number of plug-ins which are available for it. I am also most familiar with this CI engine as TeamCity (and the process outlined in this article) is one of the tools I use in my day-to-day work. A feature-comparison of Continuous Integration systems is available at [Alternatives to CruiseControl: CI Feature Matrix](#).

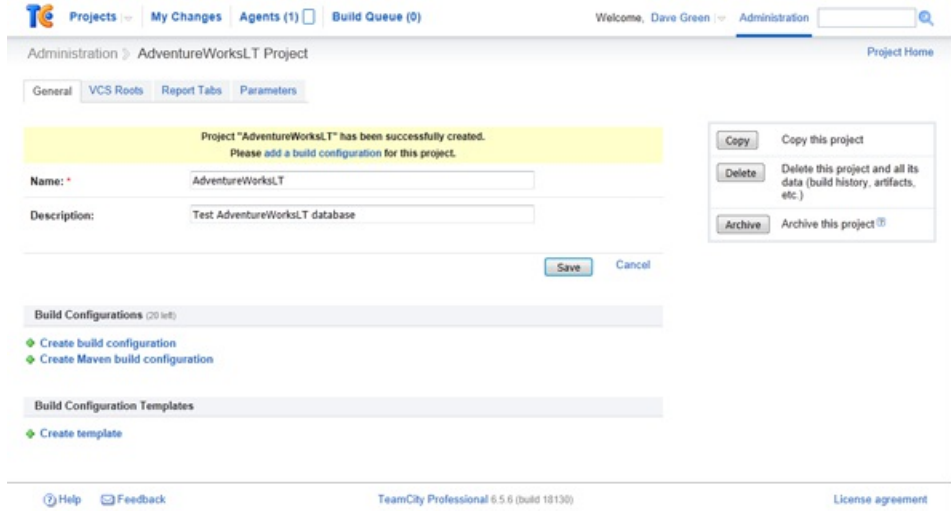
## Initial setup

My starting point for this article is a database which I have under source control using Red Gate [SQL Source Control](#), but which does not have SQL Test tests running on it. I also have a base install of TeamCity version 6.5.6. I'm using Red Gate Source Control, which plugs into SQL Server Management Studio and allows you to put database objects and data into a version-controlled environment. It's not version control itself, but rather a good interface into your existing system.

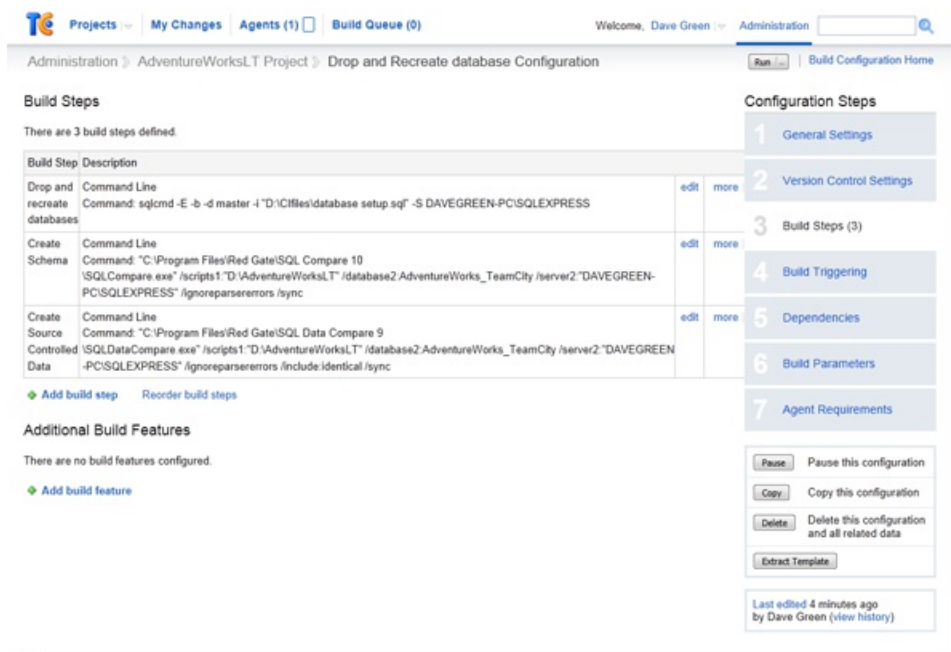
The SQL Test product is powered by [tSQLt, the TSQL-Based Test framework](#), which is free and open-source, and stores its objects in the database under test. This is great whilst the database is being developed, but this fact needs to be remembered when it comes to making sure that the tests don't make it into a shipped product. I'll explain more about that in a moment. SQL Test provides a convenient add-in to SQL Server Management Studio to help you create and run tSQLt unit tests within the design environment.

First of all, I'm going to set up TeamCity to build my database just as it is, with neither tests nor test infrastructure objects added to it. Then we will insert the call into TeamCity to run the SQL Test tests every time a build runs. This allows us to regression-test each build automatically.

I'm using the [AdventureWorksLT2008](#) sample database for this article, although you can use any database you like – I have a number of cross-database projects using this process in my work environment. The Database can be downloaded from Microsoft's website. I start by setting up a new Project in TeamCity and calling it **AdventureWorksLT**.



Then I add the build steps as shown in the following screenshots:

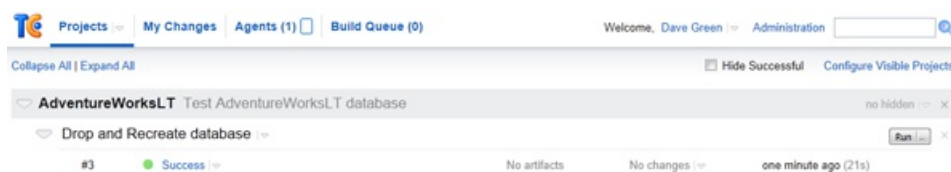


**Note:** I run through each of the build steps at the end of this article, which will explain what each of the parameters does.

Obviously you will need to change your database name and server to match. If you are using Integrated Windows authentication as I am, you will also need to ensure that the Windows account which the TeamCity Agent is using (the Agent's service account) has login access to your database server for integrated authentication, and has sufficient rights to create a database. At this point, this means the [dbcreator](#) fixed server role.

I'm using Integrated Authentication for the build steps, but you could use SQL Authentication if you want to, or even, perhaps, run the process on a remote server. The command line interface to sqlcmd will accept SQL Authentication and this is documented at [MSDN: The sqlcmd Utility](#). *Note that if you decide to use SQL Authentication, these credentials must be passed in as parameters in the build step, so would become visible to other users of the TeamCity application (including guest users if this facility is enabled).*

This should now run through OK. Let's test it by clicking Run on the build step.

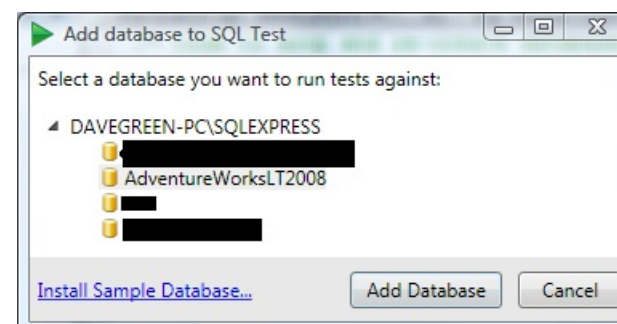


You should also be able to see what TeamCity has been doing within the Run Log, which is accessed from the drop down menu next to the run status. This is where you can find more technical information as to any failures, as it is the output (everything returned to the calling process) from the commands that you have run, and this can be examined. This is also time-stamped, so you can spot any significant time delays.

## Adding the tSQLt objects

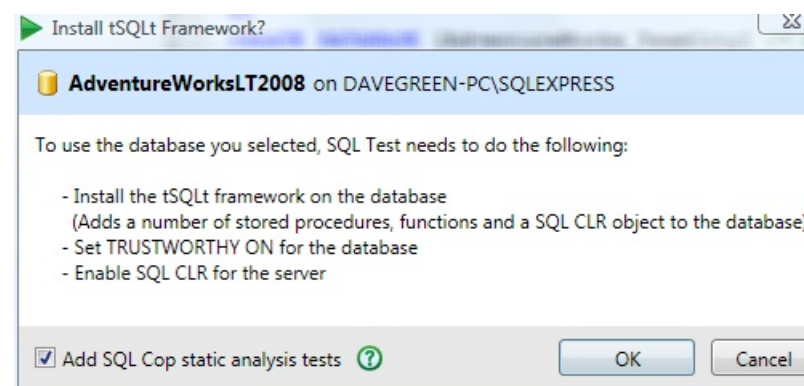
Now let's add the tSQLt objects to the database. I'm going to use the [SQLCop](#) demonstration set of tests, but of course you would want to add in your own tests. Adding the SQL Test objects to a database is done using the SQL Test window within SSMS.

Open the SQL Test window and click the link "+Add Database to SQL Test...". This will bring up a window like this (I've blacked out irrelevant databases on my SQL Server instance):



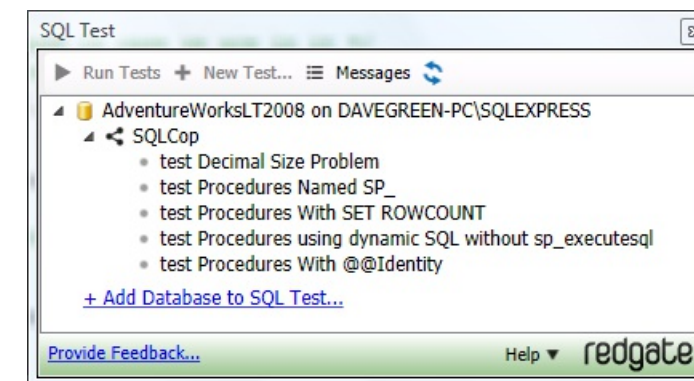
Click on 'Add Database', and select the tickbox "Add SQLCop static analysis tests", then click "OK". This will make the changes as outlined in the dialog box shown below.

The setting which enables CLR is server-wide, not database specific, and will require appropriate database server permissions (you need to have [ALTER SETTINGS](#) permission to run this step; this is implicit for members of sysadmin, or serveradmin server roles)



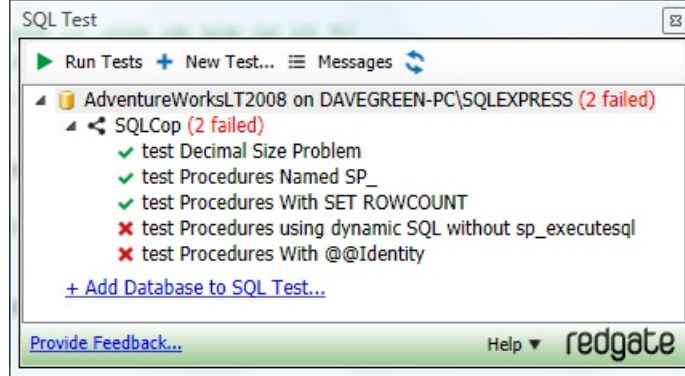
[The SQL Cop tests support best practices](#), and are useful as an example of how tests work. They are by no means the only types of tests, although custom tests are outside the scope of this article. The [tSQLt user guide](#) and [this article by the developers behind tSQLt](#), has more details about the process of creating custom tests.

Your window should now look something like this:



**Note:** I have encountered problems when trying to add the SQL Test objects to the database when the database is not owned by the sa user. You can rectify this by using the [ALTER AUTHORIZATION](#) command.

You can manually run all of the tests by clicking on the database name and then the "Run Tests" button (tests or categories can also be individually run). When we run the tests, the following is the output:



This shows us that we expect two failures - so this is what we will look out for in TeamCity, as, in our demonstration for this article, the test results shown in TeamCity should be the same as those shown by SQL Test. In practice, TeamCity would probably be set up to run other types of test such as system, integration or functional testing in a production instance of the process.

## Creating the build process

So as to ensure repeatability by providing a clean build, I use a fresh database for TeamCity testing. This is why the first step in the build process is to run a .sql file (also under source control) which drops and recreates the database. tSQLt requires that the database server is able to use CLR functions, which we can set once with the following code:

```

/* Turn on CLR on this server. This setting only needs to be changed once, as it's server level.
Note, you need to have ALTER SETTINGS permission to run this -
(this is implicit for members of sysadmin, or serveradmin server roles)
See - http://technet.microsoft.com/en-us/library/ms188787.aspx
*/
EXEC sp_configure 'show advanced options',1 ;
RECONFIGURE;
EXEC sp_configure 'clr enabled',1;
RECONFIGURE;

```

This is executed as part of the tSQLt install process, but if you use a separate database server for the CI process (as I do, to provide an isolated test environment) then you will need to execute this on the database server as well.

tSQLt also requires that the database is [TRUSTWORTHY](#), because it requires that the CLR assembly it uses has EXTERNAL\_ACCESS permission. This is not usually the case for a database freshly created, therefore before running tests on a database, I use the following code to turn these on: (This is the database setup.sql file referenced above)

```

/* Code to drop and re-create database, then set it to trustworthy.
This is used for integrating SQL Test with TeamCity.
Coded by Dave Green Jan 2012
*/
USE MASTER /* Leave the database in case we are in it */
IF EXISTS (SELECT name FROM sys.databases WHERE name = N'AdventureWorks_TeamCity')
DROP DATABASE [AdventureWorks_TeamCity]
GO
CREATE DATABASE [AdventureWorks_TeamCity] /* More options here for how you want to create the
database. */
GO
/* Set database as trustworthy for tSQLt Unit Testing CLR objects.*/
USE [AdventureWorks_TeamCity]
ALTER DATABASE [AdventureWorks_TeamCity] SET TRUSTWORTHY ON;

```

The account used to run this script will need to be a member of the fixed server role in order to mark the database as [sysadmin TRUSTWORTHY](#). If you cannot grant sysadmin permissions to this script, you could instead use a script to empty the database, and not drop/create it. This would only require [db\\_owner](#) database permissions. I'd recommend a clean database if you can though.

## Creating the schema build script

I then use Red Gate's SQL Compare product to get the schema for my database created. I'm calling it via the command line to synchronise my newly created test database (AdventureWorks\_TeamCity) with the source controlled database objects.

**Note** - you may find that on a server with a service account running TeamCity there is a failure when the product tries to generate a



message box (which, running in command line mode, it can't do). Whilst I encountered this when the product attempted to advise me if it should report back to Red Gate on my experiences, I am aware that there are a few dialogue boxes that the system can try to produce even when in command line mode. There is a registry setting you can change if you can't start the application as that user to turn off the setting - just ask Red Gate's helpful support team. Red Gate support have advised me that this is set in a registry key (HKEY\_CURRENT\_USER\Software\Red Gate\SQL Tools) as a string value called "SmartAssemblyReportUsage". It's set to True or False depending on if you want to send in feature usage to Red Gate.

## Adding static data

I then use Red Gate's SQL Data Compare to populate any data which will ship with my database. This usually includes 'static data' that it is required for the database to function, such as geographical data or other static reference data. *You don't need to use "test" data explicitly because the SQL Test system allows you to create "fake" tables and use the data you create within them with the tests.*

## Testing the build

I can run all tests using the code in RunTests.sql:

```
IF EXISTS (SELECT * FROM sys.objects WHERE OBJECT_ID = OBJECT_ID(N'[tSQLt].[RunAll]'))
AND TYPE IN (N'P',N'PC'))
BEGIN
EXECUTE [tSQLt].[RunAll]
END
```

The reason for the IF EXISTS test is in case the objects don't exist, to prevent an error. *This can be useful in the initial stages of a project, where the CI setup may be done in advance of the SQL objects and tests actually existing.* By varying the database in which this code is called (the -d parameter for sqlcmd), the same snippet can be used in different build steps. *You could explicitly put this code into the command parameter for sqlcmd as a command line query, however I prefer not to do this as I find it clutters the build screen. Keeping it separate also allows me to put my supporting files within my source control program.*

I can integrate these into TeamCity using a command line build task, and by calling SQLCMD with the -b parameter, I ensure that should a test fail, the error code which the SQL command raises will result in a failed build step.

So let's add this step to the build defined above. This will result in a build step list which looks like this:

The screenshot shows the TeamCity Build Configuration page for the 'AdventureWorksLT Project'. The main area displays a list of build steps:

Build Step	Description	edit	more
Drop and recreate databases	Command Line Command: sqlcmd -E -b -d master -I 'D:\files\database setup sql' -S DAVEGREEN-PC\SQLEXPRESS		
Create Schema	Command Line Command: "C:\Program Files\Red Gate\SQL Compare 10\SQLCompare.exe" /scripts1 "D:\AdventureWorksLT" /database2 AdventureWorks_TeamCity /server2 "DAVEGREEN-PC\SQLEXPRESS" /ignoreparsererrors /sync		
Create Data	Command Line Source Command: "C:\Program Files\Red Gate\SQL Data Compare 9\SQLDataCompare.exe" /scripts1 "D:\AdventureWorksLT" /database2 AdventureWorks_TeamCity /server2 "DAVEGREEN-PC\SQLEXPRESS" /ignoreparsererrors /includeIdentical /sync		
Run tSQLt Tests	Command Line Command: sqlcmd -E -b -d AdventureWorks_TeamCity -I 'D:\files\Run Tests sql' -S DAVEGREEN-PC\SQLEXPRESS		

Additional features include 'Add build step', 'Reorder build steps', and 'Additional Build Features' (currently none are configured). On the right, there are 'Configuration Steps' (General Settings, Version Control Settings, Build Steps (4), Build Triggering, Dependencies, Build Parameters, Agent Requirements) and control buttons (Pause, Copy, Delete, Extract Template). A status bar at the bottom indicates 'Last edited a few moments ago by Dave Green (view history)'.

When we run this build, we expect a failure because we've already seen it in the tSQLt output - and sure enough we get one, although TeamCity is not specific as to what went wrong. If you go to the Build Log (accessed from the drop down next to the build failure message), you will see that some of the tests failed - and the detail of which they are.

```

[17:52:31]: [Step 4/4] in directory: C:\TeamCity\buildAgent\work\42121fce523648d0
[17:52:32]: [Step 4/4] [SQLCop].[test Procedures using dynamic SQL without sp_executesql] failed: http://blogs.lessthandot.com
[17:52:32]: [Step 4/4] dbo.concatenated
[17:52:32]: [Step 4/4] -----
[17:52:32]: [Step 4/4] dbo.uspLogError
[17:52:32]: [Step 4/4] -----
[17:52:32]: [Step 4/4] [SQLCop].[test Procedures With @@Identity] failed: http://wiki.lessthandot.com/index.php/6_Different_Wa
[17:52:32]: [Step 4/4] dbo.uspLogError
[17:52:32]: [Step 4/4] -----
[17:52:32]: [Step 4/4] +-----+
[17:52:32]: [Step 4/4] |Test Execution Summary|
[17:52:32]: [Step 4/4] +-----+
[17:52:32]: [Step 4/4]
[17:52:32]: [Step 4/4] |No|Test Case Name |Result |
[17:52:32]: [Step 4/4] +-----+
[17:52:32]: [Step 4/4] |1 |[SQLCop].[test Decimal Size Problem] |Success|
[17:52:32]: [Step 4/4] |2 |[SQLCop].[test Procedures Named SP_] |Success|
[17:52:32]: [Step 4/4] |3 |[SQLCop].[test Procedures With SET ROWCOUNT] |Success|
[17:52:32]: [Step 4/4] |4 |[SQLCop].[test Procedures using dynamic SQL without sp_executesql]|Failure|
[17:52:32]: [Step 4/4] |5 |[SQLCop].[test Procedures With @@Identity] |Failure|
[17:52:32]: [Step 4/4] -----
[17:52:32]: [Step 4/4] Msg 50000, Level 16, State 10, Server DAVEGREEN-PC\SQLEXPRESS, Line 1
[17:52:32]: [Step 4/4] Test Case Summary: 5 test case(s) executed, 3 succeeded, 2 failed, 0 errored.
[17:52:32]: [Step 4/4] -----
[17:52:32]: [Step 4/4] Process exited with code 1
[17:52:32]: [Step 4/4] Step Run tSQLt Tests (Command Line) failed
[17:52:32]: Publishing internal artifacts
[17:52:32]: [Publishing internal artifacts] Sending build.finish.properties.gz file
[17:52:33]: Build finished

```

This isn't that nice however, as it is a two step process to see what failed, and if you have a number of tests it is not very easy to pinpoint what went wrong. The output messages are also somewhat separated from the success or otherwise of the test.

There is a way however to get tSQLt to feed its test results directly into the TeamCity tests tab. This is by using the JUnit XML format to capture the output from tSQLt.

If you click on the "Add build feature" box at the bottom of the build steps window, you will see a pop up window appear. Select the feature "XML Report Processing" and "Ant JUnit" as the report type, and you will be able to type in a location for your XML file. This is where TeamCity will read the test results from during a build run.

**Note:** you can use wildcards here to denote multiple files, and potentially read in tests from a number of databases. This is useful if your project is spanning multiple databases all with unit tests in them.

You can use defined variables within the Monitoring Rules window; this can allow you to reference files within your build directory that may only be created on checkout.

*Note, I have ticked the "Verbose output" option in the build feature because I have had some issues with the tests not being properly picked up from the XML file if this is not ticked.*

We must also add a step to generate the XML, and then remove the -b from the step which runs the tests, as we do not now want the build to stop at this point in the case of a test failure (we will now record that via the XML). I do suggest leaving the -b on the step to generate the XML file though - as this adds protection in the case that something goes wrong with the XML file generation.

I've added the "Get Test Results" step in the below screenshot.

Note, you can create this step first if you prefer - there's no requirement to put in the Additional build feature first.

The screenshot shows the configuration page for a build configuration named "Drop and Recreate database Configuration". It is divided into two main sections: "Build Steps" and "Additional Build Features".

**Build Steps:** There are 5 build steps defined. The "Get Test Results" step is highlighted in blue. The steps are:

Build Step	Description	edit	more
Drop and recreate databases	Command Line Command: sqcmd -E -b -d master -I "D:\Cifiles\database setup.sql" -S DAVEGREEN-PC\SQLEXPRESS		
Create Schema	Command Line Command: "C:\Program Files\Red Gate\SQL Compare 10\SQLCompare.exe" /scripts1 "D:\AdventureWorksLT" /database2 AdventureWorks_TeamCity /server2 "DAVEGREEN-PC\SQLEXPRESS" /ignoreparserserrors /sync		
Create Source Data	Command Line Command: "C:\Program Files\Red Gate\SQL Data Compare 9\SQLDataCompare.exe" /scripts1 "D:\AdventureWorksLT" /database2 AdventureWorks_TeamCity /server2 "DAVEGREEN-PC\SQLEXPRESS" /ignoreparserserrors /include.identical /sync		
Run SQL Tests	Command Line Command: sqcmd -E -d AdventureWorks_TeamCity -I "D:\Cifiles\Run Tests.sql" -S DAVEGREEN-PC\SQLEXPRESS		
Get Test Results	Command Line Command: sqcmd -E -b -S DAVEGREEN-PC\SQLEXPRESS -d AdventureWorks_TeamCity -h -1 -y0 -I -I "D:\Cifiles\GetTestResults.sql" -o "D:\Cifiles\TestResults.xml"		

**Additional Build Features:** There is 1 build feature configured.

Feature Type	Parameters	Description	edit	delete
XML report processing	Import Ant JUnit reports from	D:\Cifiles\*.xml		

**Configuration Steps:** A sidebar on the right lists configuration steps: 1 General Settings, 2 Version Control Settings, 3 Build Steps (5), 4 Build Triggering, 5 Dependencies, 6 Build Parameters, 7 Agent Requirements. Below this are buttons for "Pause", "Copy", "Delete", and "Extract Template".

If we run through again, we still get a failure message, but we now get a list of which tests they are, and the ability to drill into them for more information. This is much more helpful for a developer team.

The screenshot shows the build results page for the "Drop and Recreate database" build. The build is marked as failed (#7) with 2 tests failed and 3 passed. The "Tests" tab is selected, showing a list of test results:

Status	Test	Duration	Order#
OK	SQLCop. test Decimal Size Problem	< 1ms	1
OK	SQLCop. test Procedures Named SP	< 1ms	2
Failure	SQLCop. test Procedures using dynamic SQL without sp_executesql	< 1ms	3
Failure	SQLCop. test Procedures With @@Identity	< 1ms	4
OK	SQLCop. test Procedures With SET ROWCOUNT	< 1ms	5

By using the XML result processing, you can use TeamCity's built in Mute / Investigate test options, to better manage your build process, and prevent a known failure stopping a build.

The screenshot shows the build results page for the "Drop and Recreate database" build. The build is now marked as passed (#8) with 3 tests passed and 2 muted. The "Tests" tab is selected, showing a list of test results:

Status	Test	Duration	Order#
OK	SQLCop. test Decimal Size Problem	< 1ms	1
OK	SQLCop. test Procedures Named SP	< 1ms	2
Muted	SQLCop. test Procedures using dynamic SQL without sp_executesql	< 1ms	3
Muted	SQLCop. test Procedures With @@Identity	< 1ms	4
OK	SQLCop. test Procedures With SET ROWCOUNT	< 1ms	5

The run log now looks like :

```

Overview | Changes (0) | Tests | Build Log | Build Parameters | All History | Last recorded build
Important messages | All messages | Tree view | Tail | Download full build log (-27.96Kb) | zipped
[18:28:31]: [Step 4/5] 14 | [SQLCop].[test Procedures using dynamic SQL without sp_executesql]|Failure!
[18:28:31]: [Step 4/5] 15 | [SQLCop].[test Procedures With @@Identity] |Failure!
[18:28:31]: [Step 4/5] -----
[18:28:31]: [Step 4/5] Msg 50000, Level 16, State 10, Server DAVEGREEN-PC\SQLEXPRESS, Line 1
[18:28:31]: [Step 4/5] Test Case Summary: 5 test case(s) executed, 3 succeeded, 2 failed, 0 errored.
[18:28:31]: [Step 4/5] -----
[18:28:31]: [Step 4/5] Process exited with code 0
[18:28:31]: Step 5/5: Get Test Results (Command Line)
[18:28:31]: [Step 5/5] "sqlcmd" is not present in directory C:\TeamCity\buildAgent\work\42121fce523648d0
[18:28:31]: [Step 5/5] Starting: C:\Windows\system32\cmd.exe /c sqlcmd -E -b -S DAVEGREEN-PC\SQLEXPRESS -d AdventureWorks_Team
[18:28:31]: [Step 5/5] in directory: C:\TeamCity\buildAgent\work\42121fce523648d0
[18:28:31]: [Step 5/5] Process exited with code 0
[18:28:32]: [Step 5/5] SQLCop
[18:28:32]: [SQLCop] test Decimal Size Problem
[18:28:32]: [SQLCop] test Procedures Named SP_
[18:28:32]: [SQLCop] test Procedures using dynamic SQL without sp_executesql
[18:28:32]: [test Procedures using dynamic SQL without sp_executesql] http://blogs.lessthandot.com/index.php/DataMgmt/DataDe
dbo.concatenated
[18:28:32]: [SQLCop] test Procedures With @@Identity
[18:28:32]: [test Procedures With @@Identity] http://wiki.lessthandot.com/index.php/6_Different_Ways_To_Get_The_Current_Iden
dbo.uspLogError
[18:28:32]: [SQLCop] test Procedures With SET ROWCOUNT
[18:28:32]: [Step 5/5] D:\Cifiles\TestResults.xml report processed: 1 suite, 5 tests
[18:28:32]: Ant JUnit report watcher
[18:28:32]: [Ant JUnit report watcher] 1 report found for paths:
[18:28:32]: [Ant JUnit report watcher] D:\Cifiles\*.xml
[18:28:32]: [Ant JUnit report watcher] Successfully parsed
[18:28:32]: [Successfully parsed] 1 report
[18:28:32]: [Successfully parsed] D:\Cifiles\TestResults.xml
[18:28:32]: Publishing internal artifacts
[18:28:32]: [Publishing internal artifacts] Sending build.finish.properties.gz file
[18:28:32]: Build finished

```

We can see that step 5/5 still shows Red text (which usually indicates failure) for the specific step. This is because tests failed, however because the failing tests are muted the build is successful (Green). The individual tests that failed are also shown in Red here, together with the failure message that the test produced.

## Creating the deployment package

Because I plan to use [SQL Packager](#) to package my code, I must delete all tests and the tSQLt objects as my next step. I do this with the following code:

```

/* This is the clear-up script for Red Gate SQL Test */
/* Coded Dave Green 2012-01-11.*/
--DROP ALL tests
IF EXISTS (SELECT * FROM sys.views WHERE OBJECT_ID = OBJECT_ID(N'[tSQLt].[TestClasses]'))
BEGIN
DECLARE @TestClassName VARCHAR(MAX)
DECLARE tests CURSOR LOCAL FAST_FORWARD FOR
SELECT Name FROM tSQLt.TestClasses;
OPEN tests;
--Cycle through each test class and drop them (you have to do this one at a time).
FETCH NEXT FROM tests INTO @TestClassName;
WHILE @@FETCH_STATUS = 0
BEGIN
EXECUTE [tSQLt].[DropClass] @TestClassName
FETCH NEXT FROM tests INTO @TestClassName;
END;
CLOSE tests;
DEALLOCATE tests;
PRINT 'Tests Dropped'
END
ELSE
BEGIN
PRINT 'No Tests Found.'
END
--Drop test Harness - this actually removes the tSQLt objects.
IF EXISTS (SELECT * FROM sys.objects WHERE OBJECT_ID = OBJECT_ID(N'[tSQLt].[Uninstall]') AND TYPE
IN (N'P', N'PC'))
BEGIN
EXEC [tSQLt].[Uninstall]
PRINT 'Test System Dropped'
END
ELSE
BEGIN
PRINT 'Test System Not Found.'
END
END

```



You may not need to do this if your build process does not require you to create a deployment package, but it can be useful to clean up the database anyway, if only to ensure that the procedures which do this clear-up are tested.

I would then call SQL Packager as the final build step to package the database. This is outside the scope of this article, however information on how to call SQL Packager from the command line can be found [on the Red Gate website](#).

## Conclusion

You now have a continuous build engine which will be able to build from source control and run your unit tests against them. By including schema-based checks, for example those by SQLCop, you can ensure that your development project keeps up with best practice and also conventions within your organisation.

I would suggest that you work out a test naming scheme at this stage and keep to it - particularly for a larger development team this will help prevent 'losing' tests, and to make it easier to understanding of their purpose.

The approach taken by SQL Test and tSQLt in putting the tests in as database objects can be a little contentious, but is helpful in that the test versions are in the same source control as the main database, thus helping promote tests being modified at the same time as, or in advance of, code development. There is also more chance that tests will be updated and run with code changes if a CI environment is used.

You can use TeamCity's ability to integrate with your version control system to allow you to trigger builds for code changes in particular projects, and even automatically promote / deploy working code. This can include [migration scripts](#) for updating live environments to avoid potential data loss. This is outside the scope of this article, but a variety of useful information can be found on the TeamCity and Red Gate websites.

I found the process to be fairly painless; and whilst the SQL Test program is currently in preview, it could do with a little work on the process to remove test classes for use in a packaged CI environment. I've been very impressed with TeamCity for CI use, and will certainly be using it for on-going development. Overall I am very positive with the process I have demonstrated in this article, as it is a relatively simple to implement solution which helps ensure a higher level of testing and code quality with little subsequent effort for developers.

*It's worth mentioning that I have separately used this process with [AccuRev](#) as a source control system, and this integrates well with TeamCity allowing you to trigger builds from source control with a simple plugin.*

## Appendix

### A note on security for TeamCity setup

Whilst I am by no means a security expert, I have thought about the set-up for TeamCity, and the required permissions for using a process such as that outlined above. In my opinion, best practice is that the service should run with the least necessary permissions.

In this article, I have TeamCity running as a local system account, but in my CI environment I use a dedicated network account which has the permissions necessary (sysadmin on the database server to mark DBs as TRUSTWORTHY, and read/write to the file locations under test, and those for the TeamCity agent work directory). The account will also need "log on as a service" rights, in order that TeamCity can run its Agent, as well as the [permissions listed in the TeamCity documentation](#). That being said, it's not a (Windows) server administrator, nor does it have many permissions on the wider domain. There may optionally be something required (for example file permissions) if you have the TeamCity installation tied into version control using integrated authentication.

In practice, I would try to give the CI system its own database server to use, as I don't want unchecked (i.e. pre-test-pass) code on a UAT (or production!) server, and I would like to keep it separate from the server on which the code is developed. Your environment will dictate this somewhat.

You might even use [different agents on the same server](#) running as different users to segregate permissions. For example, Agent 1 has permissions on local database and runs the job as it currently stands. Agent 2 has permissions on the UAT server, and can only run a new final build step (which Agent 1 can't run), which deploys the objects to a UAT server. This way, the permissions are isolated to what is required for each environment.

I prefer not to run using SQL Authentication as this means that the credentials are stored within the TeamCity server, and so somewhat more exposed than if we use integrated security. (Who has access to your TeamCity server / configuration?)

## Build step breakdown

### Step 1 - Drop and recreate databases

```
sqlcmd -E -b -d master -i "D:\Cifiles\database setup.sql" -S DAVEGREEN-PC\SQLEXPRESS
```

This step uses sqlcmd (a command line SQL client which is included in SQL Server) to talk to the server DAVEGREEN-PC\SQLEXPRESS (specified with -S parameter), starting in the database master (-d parameter).

Sqlcmd is set to use integrated authentication (-E parameter) to connect, and runs the contents of the file "D:\Cifiles\database setup.sql" (-i parameter). The -b parameter means that it will cause an error condition to be returned if a SQL error is raised, which will in turn cause a build failure.

The script file (which is listed in the article above) will drop the database, re-create it, and set it to be TRUSTWORTHY.

## Step 2 - Create Schema

```
"C:\Program Files\Red Gate\SQL Compare 10\SQLCompare.exe" /scripts1:"D:\AdventureWorksLT"
/database2:AdventureWorks_TeamCity /server2:"DAVEGREEN-PC\SQLEXPRESS" /ignoreparsererrors /sync
```

I'm using Red Gate's SQL Compare product to compare the scripts in "D:\AdventureWorksLT" with the AdventureWorks\_TeamCity database on my SQL instance, "DAVEGREEN-PC\SQLEXPRESS". The 1 and 2 prefixes relate to the "from" and "to" portions – this means you could compare 2 databases. I have specified the /sync parameter to instruct SQL Compare to make changes to my "2" database, and I've specified the /ignoreparsererrors parameter because I was initially getting some issues with source controlled data causing parser failures.

## Step 3 - Create Source Controlled Data

```
"C:\Program Files\Red Gate\SQL Data Compare 9\SQLDataCompare.exe" /scripts1:"D:\AdventureWorksLT"
/database2:AdventureWorks_TeamCity
/server2:"DAVEGREEN-PC\SQLEXPRESS" /ignoreparsererrors /include:identical /sync
```

Here the SQL Data Compare product is used to compare the data within the source control repository with that in the newly created database. Whilst most parameters are the same as in step 2, the addition of the /include:identical parameter will ensure that only objects which exist in both databases are copied. Note, you can use an XML file to drive SQL Compare and SQL Data Compare if you wish to, which keeps security and other parameters from being exposed within the TeamCity interface, at the cost of making the process a little less clear. Using an XML file is also a way of restricting comparisons to a specific list of tables, and I've used it as a way of reading specific tables from another environment which may have more tables in common than I wish to copy across.

## Step 4 - Run tSQLt Tests

```
sqlcmd -E -d AdventureWorks_TeamCity -i "D:\Cifiles\Run Tests.sql" -S DAVEGREEN-PC\SQLEXPRESS
```

Again we are using sqlcmd, this time to run a script which runs all the tSQLt tests. Note, we have not put the -b parameter here, since we don't want the execution to stop if an error is detected in a test. This is because this is handled in the next step:

## Step 5 - Get Test Results

```
sqlcmd -E -b -S DAVEGREEN-PC\SQLEXPRESS -d AdventureWorks_TeamCity -h-1 -y0 -I -i
"D:\Cifiles\GetTestResults.sql" -o "D:\Cifiles\TestResults.xml"
```

We're now using sqlcmd (with it's -b parameter to fail on error) to retrieve the test results, and output these to an XML file which the Ant JUnit XML Report Processing feature will pick up. The other parameters -h-1 -y0 -I allow the XML to be output cleanly.

## References and Acknowledgements:

- SQL Test - <http://www.red-gate.com/products/sql-development/sql-test/>
- TeamCity - <http://www.jetbrains.com/teamcity/>
- An article on unit testing with tSQLt - <http://www.simple-talk.com/sql/t-sql-programming/sql-server-unit-testing-with-tsqlt/>
- The user guide for tSQLt - <http://tsqlt.org/user-guide/>
- Troy Hunt's excellent articles on TeamCity integration - [Continuous Integration for SQL Server Databases](#) and [TeamCity, Subversion & Web Deploy part 1: Config transforms](#)



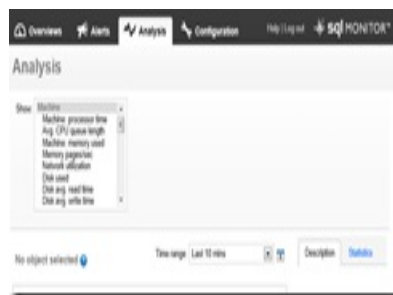
# Tuning Red Gate: #1 of Many

Published Thursday, February 09, 2012 4:10 PM

Everyone runs into performance issues at some point. Same thing goes for Red Gate software. Some of our internal systems were running into some serious bottlenecks. It just so happens that we have this nice little SQL Server monitoring tool. What if I were to, oh, I don't know, use the monitoring tool to identify the bottlenecks, figure out the causes and then apply a fix (where possible) and then start the whole thing all over again? Just a crazy thought.

OK, I was asked to. This is my first time looking through these servers, so here's how I'd go about using SQL Monitor to get a quick health check, sort of like checking the vitals on a patient.

First time opening up our internal SQL Monitor instance and I was greeted with this:



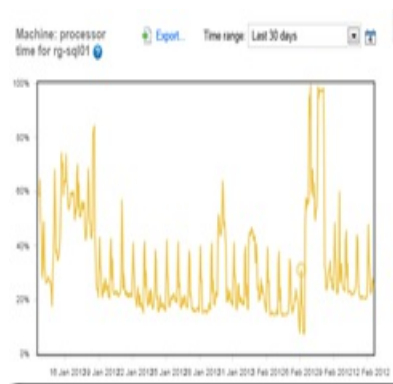
Oh my. Maybe I need to get our internal guys to [read my blog](#).

Anyway, I know that there are two servers where most of the load is. I'll drill down on the first. I'm selecting the server, not the instance, by clicking on the server name. That opens up the Global Overview page for the server. The information here much more applicable to the "oh my gosh, I have a problem now" type of monitoring.



But, looking at this, I am seeing something immediately. There are four(4) drives on the system. The C:\ has an average read time of 16.9ms, more than double the others. Is that a problem? Not sure, but it's something I'll look at. It's write time is higher too.

I'll keep drilling down, first, to the unclosed alerts on the server. Now things get interesting. SQL Monitor has a number of different types of alerts, some related to error states, others to service status, and then some related to performance. Guess what I'm seeing a bunch of right here:



Long running queries and long job durations. If you check the dates, they're all recent, within the last 24 hours. If they had just been old, uncleared alerts, I wouldn't be that concerned. But with all these, all performance related, and all in the last 24 hours, yeah, I'm concerned.

At this point, I could just start responding to the Alerts. If I click on one of the the Long-running query alerts, I'll get all kinds of cool data that can help me determine why the query ran long. But, I'm not in a reactive mode here yet. I'm still gathering data, trying to understand how the server works. I have the information that we're generating a lot of performance alerts, let's sock that away for the moment.

Instead, I'm going to back up and look at the Global Overview for the SQL Instance. It shows all the databases on the server and their status. Then it shows a number of basic metrics about the SQL Server instance, again for that "what's happening now" view or things. Then, down at the bottom, there is the Top 10 expensive queries list:



This is great stuff. And no, not because I can see the top queries for the last 5 minutes, but because I can adjust that out 3 days. Now I can see where some serious pain is occurring over the last few days. Databases have been blocked out to protect the guilty.

That's it for the moment. I have enough knowledge of what's going on in the system that I can start to try to figure out why the system is running slowly. But, I want to look a little more at some historical data, to understand better how this server is behaving. More next time.

by [Grant Fritchey](#)  
Filed Under: [Monitoring](#)

# How to Kill a Company in One Step or Save it in Three

07 February 2012

by Wesley David

The majority of companies that suffer a major data loss subsequently go out of business. Wesley David remembers vividly the day when the organisation he worked for found that they couldn't restore their data, and the subsequent struggles that ensued. Shoulda-woulda-coulda.

Once upon a time, when I was knee high to a help desk position, I witnessed a near fatality. Not of a person, but an organization. An entire business was nearly hewn down by the sickle of shoulda-woulda-coulda.

The organization that I was working for was growing rapidly and stretching the practical limits of its suite of off-the-shelf customer relationship management applications. One person was solely responsible for any application customizations using .NET and tweaking the SQL Server database that the CRM package used.

The database server infrastructure was typical for a small to medium sized business. There were only two independent SQL Servers that served the entire organization's RDBMS needs. The reason for having two was simply to split the workload of multiple applications and not for any data redundancy or failover purposes. The servers were curated by one SysAdmin who was spread out over everything that plugged into an electrical outlet and one network admin who handled anything that touched the internet.

The developer responsible for developing and modifying the CRM package had to constantly be working with the database, documenting its idiosyncrasies, and tweaking it to meet our expanding needs. That database was a maze of twisty relationships (and not just of the foreign key kind either, but I'm digressing too far) and none of them were alike. It was a tedious and often frustrating job.

Of course, when working on changes to the application and its data, a test database was used. It was refreshed to contain the latest information from the production database every so often. It wasn't important to keep test data completely bleeding edge. As long as the table structure was the same as the live database, things were great.

One tedious day, a certain DML statement was run against the test database. The statement was required in order to shift the structure of the database for a new workflow. We could argue over if that practice is ever a good idea, but I'm not in the mood to weep uncontrollably. That DML statement apparently carried with it some poor logic because it dropped or otherwise destroyed *every table in the database*. It was a spectacular event; a morbidly beautiful performance. Once the rush of fascination ebbed, however, it was time to reload the test database and try again.

Except, after the developer did a double-take and engaged in a few seconds of eye-rubbing, he saw that the test database was perfectly fine.

*Ring! Ring!*

Desks phones that ring seconds after a destructive operation is performed are to be noted as telling signs. "Is the server down?" asked one of our chipper customer service reps.

In the course of my career, I have discovered some visual indicators that almost universally spell trouble in an IT department. The first is smoke. The second is flashing lights next to a container of halon. The third, and arguably most dangerous, is a developer in a server room. Especially if the developer is asking about backups.

Within a minute of the fateful DML statement being executed, the developer, Sr. Systems Administrator and our Network Engineer had converged in the server room. To the credit of that IT department, no one was panicking. No one was shouting. Not one accusatory or angry word was ever heard during or after the ordeal. We had problems to solve.

## Defeat Snatched from the Jaws of Victory

"Not a problem! We've got backups!" the Systems Administrator said. I watched over his shoulder as he pulled out the rack monitor, clicked a button on the KVM switch and logged in to the SQL Server in question. Of course we had backups! Of course they were automated! Of course things would be fine! This situation would undoubtedly be but a brief recess in the day's scheduled activities. He investigated the SQL Server instance to determine what the recovery process would be (no, we didn't have written, much less *practiced*, disaster recovery documentation).

It was then that the School Marm of Swift Kicks to the Posterior mordantly instructed the IT department in the ignobility of simple recovery mode. Yes, when we checked the settings of the database we noticed that it was set to simple recovery mode. That meant that the database's transaction logs could not be replayed into the main database file once it was recovered. At that point we were therefore assured of losing all transactions since the last backup which had been taken the night before. A morning's worth of customer interaction was gone. We were abashed, but at least things could have been worse.

(As a side note, understand that you're already in a bad situation if you're finding solace in comparing the current state of affairs with theoretically worse situations. Make special note if the only theoretical situations you can think of involve meteors, aliens and/or bubonic plague.)

It was then that to the School Marm's injury, insult was added. Some fellow named Murphy barged into the situation and started laughing maniacally while pointing to the backup archives. Each one of the daily backups that we had on hand was inconsistent. Even worse, we only had a handful of

archives to check because only a few days' worth of daily database backups were retained as part of the backup job.

This part of the story involves some hand-waving because I don't believe anyone ever figured out what was wrong with the backups that had been taken. Apparently the size of the backups was fine, so *some* kind of data existed. It wasn't the old "the-backups-are-really-just-symbolic-links!" routine that I'm familiar with. We weren't using any fancy backup program that had been modified awry. We simply used the SQL Server Agent to take nightly backups and place them on a file share. Perhaps the corruption was due to recent volume moves as we settled into our new SAN. The SAN freed up more space on NAS and DAS appliances so files and volumes were being shuffled hither and thither. Perhaps it was a bad RAID card trashing files or a very, very unfortunately located portion of bad sectors. No one knows for sure. It was truly bizarre and exemplifies that backups are nothing without verifying them through proper restoration practice.

Each pair of pupils in the server room dilated to specks as we realized what this meant. We had no viable backups. There were no means of returning to a consistent database. The only options that remained were pursuing a painstaking, low level reconstruction from the oddly inconsistent database backups or immediately shutting down the server, removing the hard drives, cloning them and shipping the RAID set to a data recovery service to hopefully find the filesystem where the pre-mashed database once resided. Neither option was anywhere near a sure path to recovery. Both options were frighteningly expensive, both in terms of the procedures themselves and the time it would take to complete thus shutting the business down for a week or more.

"What am I witnessing?" I asked in my mind as the drone of servers filled the still air. I occasionally noticed people peeking through the 6-inch wide pane of clear glass that ran next to the full height of the server room's locked doors. They almost appeared to have thought bubbles over their heads. "What are they doing in there? Why don't they just reboot the server already?"

This was not a fly-by-night company. The organization made 8 figures in USD revenue per year and had tens of thousands of customers in nearly every continent and country across its 15 year history. As mentioned before, we were growing rapidly and trying to do our level best to manage the growth. However, a "best effort" at managing the systems was meaningless if such a chain of events had been allowed to happen. The tone in the server room made a state funeral seem airy. We were the first witnesses to the killing of an organization and possibly a few careers. It was an eerie feeling to see such a stark reality being realized and know that on the other side of the walls was a few hundred people completely unaware that they might be unemployed in a short time.

## Victory is a Dish Best Served While Still Employed

"The SAN!" the SysAdmin shouted and flicked across the server room floor like he had been galvanized. Only months earlier we had implemented the organization's first SAN, having recently grown to the point where DAS and NAS was impractical for several of our services. One of those services was our SQL Servers. iSCSI LUNs had been carved out and presented to the database servers for our most precious and I/O intensive databases to be stored on.

The SAN had some fancy features that we had all been fawning over for weeks. One of those fancy features was periodic volume snapshots. Those snapshots could be explored for files. However, could a SAN snapshot provide a consistent copy of a SQL Server database? We weren't sure.

After searching for and extricating the most recent data files from the volume snapshot, the database was mounted. DBCC CHECKDB plowed through the data. No errors were returned. Some quick SQL queries were made to pull data that the developer knew had recently been added. The data seemed to all be there and in fine form. After a bit more exploring, we were satisfied that the snapshot had saved us from losing all but the last few hours of data. It was actually a better recovery than had one of the daily backups been consistent and mounted. We reconnected the database, got our users back onto their normal daily work and began the self-effacing process of learning from our mistakes.

## The Light at the End of the Tunnel Wasn't a Train

Fortunately, the organization survived and so did everyone's jobs. Very few people ever knew how close to ultimate disaster the organization came. The leadership was gracious, knowing how crazy the growth had been and that nearly every department was running on the fumes of energy drinks in order to keep up. They simply expected the IT department to change whatever was necessary to prevent the same mistakes from happening ever again. There was an awful lot to learn from this situation. Here are three major points to take away from this so that you can avoid a terrible fate.

**First**, treat production data like the precious thing that it is. It is the family jewels, the family house and the family itself. Without it, there is nothing. Protect it jealously. Separate your development environment as far as possible from your production environment. That means separate servers, separate networks, separate accounts, separate everything.

In the course of working on your test environment, perform your actions with user accounts that do not have access to production resources. In fact, explicitly deny those user accounts access. Make the exportation of production data as safe, fresh, automated and one-way as possible so that, even if there's no technical possibility of accidentally using production systems, there is never a temptation to intentionally allow an untested operation "just this once."

**Second**, backup your data like you know you should. Put the effort into getting things right and understanding what types of backups you are taking. Are you taking full backups once a day? Differential or incremental? *Both* differential and incremental? Are database snapshots being taken and if so what level of backup is that offering? Are volume snapshots being taken by the storage system and if so do they support snapshots of your specific RDBMS files? Do you need CDP level backups? Is there a cluster involved and if so what impact does its architecture have on the backups being taken?

Let's pause here for a moment and discuss the confectionary bit of technology that saved the day: SAN volume snapshots. If you're fortunate enough to have your databases on a SAN volume, consider that you may have some extra threads in your safety net, but that's about all you can

take comfort in. SAN snapshots are very proprietary things and may or may not support the consistent restoration of your RDBMSs data files. They also might not be turned on and if they are, you need to pay particular attention to things like what level of point-in-time recovery is allowed, how much history is preserved and if the space that snapshots take up is dynamically allocated (thus allowing for a sudden deletion if space is needed elsewhere). In other words: SAN snapshots are not proper backups. Even if they turn out to be useful, you must pay special attention to their sundry options and caveats. They are a tasty morsel that can be snacked on if the situation calls for it, but nothing more.

There are virtually innumerable enumerations that one could consider when inspecting their systems' backup procedures. However, the questions above cover a lot of ground and will get you thinking in the right direction. Just don't stop with those *questions*. Make sure you're actually *taking* backups! Backup early, backup often. Nevertheless, even *that* is insufficient, because you must...

**Thirdly**, test your backups! In fact, stop referring to your backups as "backups" and refer to them as "restores." If you haven't actually verified and restored your backups, then refer to them as "alleged restores." You must go through the entire steps of a full restoration procedure to be certain of their viability. Certainly you can automate parts of a restoration test. In fact, that automation can come in handy in the event of actually restoring from a disaster. Just make sure to get your hands on the test restoration process as well. Document it thoroughly in minute detail. Have others review your documentation to make sure it's cogent.

I recently encountered an organization that performs a full restoration procedure for their systems every six month. They even throw parties when things go well. Fortunately, their last restoration drill resulted in much pizza and confetti. The same can rarely be said for most organizations.

Does all of the above sound like a lot of work? *It is* a lot of work. Work for which you will be paid. Work for which you will be in danger of being out of if you neglect data protection. There are only two or three things over which someone in the IT field can completely ruin their career. One of them is carelessly bungling company data through improper backups. Word gets around and the smoke of that failure clings to a person's garments for years to come.

Consider today what you can do to safeguard your company and your career from backup-borne catastrophe. If you're overworked and not being enabled to pursue a data protection initiative, shine up your curriculum vitae and go job hunting because you're, in essence, being told to sit on a bomb and hope it doesn't do you harm.

Do you have a similar experience? Can you find other lessons to be learned from the situation I've just shared? I know of a few that I didn't form into words. Undoubtedly this anecdote could be turned into quite a series on safeguarding data and implementing proper IT procedures (as well as how colleagues should treat each other; sincerely, everyone involved behaved as a perfect gentleman throughout the ordeal). Are you impeded from properly backing up and testing restores? Do you seem to be unable to communicate the dangers that exist? Cry on my shoulder or shout into my ears in the comments below.





# TortoiseSVN and Subversion Cookbook Part 4: Sharing Common Code

13 February 2012  
by Michael Sorens

Michael Sorens continues his series on Source Control with Subversion and TortoiseSVN by describing several ways one can use to share code among several projects.

This is the fourth installment of the TortoiseSVN and Subversion Cookbook series, which is a collection of practical recipes to help you manage source control with Subversion and its ubiquitous GUI front-end, TortoiseSVN. So far this series has covered:

- [Part 1](#): Checkouts and commits in a multiple-user environment.
- [Part 2](#): Adding, deleting, moving, and renaming files, plus filtering what you add.
- [Part 3](#): Putting things in and taking things out of source control.

This installment explains the simple and elegant mechanism for sharing code among several projects, with both the separate projects and the shared code under source control.

*Reminder: Refer to the [Subversion book](#) and the [TortoiseSVN book](#) for further information, and as directed in the recipes below.*

Sharing common code in source control sounds daunting. In Subversion it really is simple—once you have seen how to do it. So I expect that, once you read this recipe, you will likely never need to come back to it; the whole concept will be folded into your “That’s obvious!” mental bucket.

To create linkages to external/shared code you need an appreciation of the following sometimes subtle points:

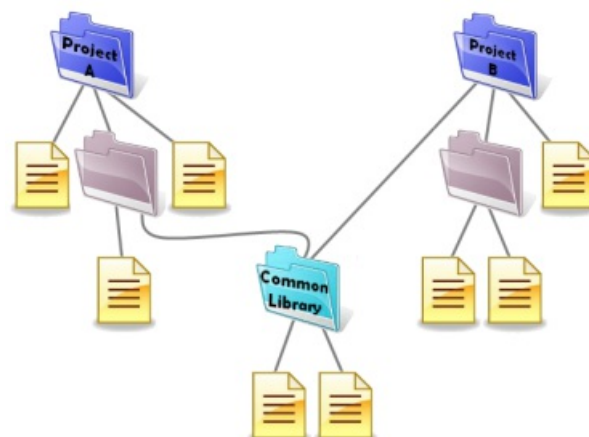
- You need to decide whether you want to use an absolute reference to the shared code repository or a relative reference. An absolute reference is simpler on the face of it, but introduces maintenance issues. If you choose instead to use a relative reference, you need to know the location of the shared code repository in relation to your local code. Depending on the distance between the two (which you will understand shortly) you can select one of the four relative addressing notations, or even a combination of them.
- You need to decide whether to use the head revision of the shared code repository or an explicit revision. The former is again, on the face of it simpler, but should almost never be done as you will soon learn.

A linkage involves specifying where you want to connect shared code into your local tree. You can make that specification right at the connection point itself but you are not limited to that point—you can actually specify the connection elsewhere (e.g. at the root of your project)!

The `svn:externals` property is the nexus of all things relevant to sharing code. With Subversion 1.7 it is easier than ever to set the fields of this property to properly specify your local connection point, your external reference, and the appropriate revision.

## Reusing common code

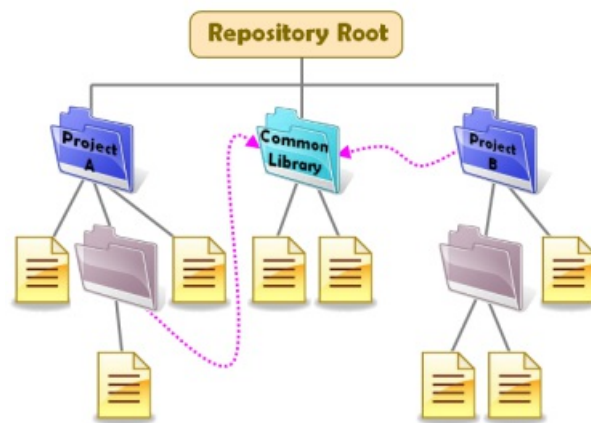
Inevitably, when you have several projects under source control you will want to avoid duplication of code by sharing some code between two or more of them. Subversion provides a facility for linking projects in order to make this possible. In Figure 4-1, for example, you have two projects sharing some common code. This is quite simple in concept, but since they are separate projects in your Subversion repository, how then can they both include the common library under source control?



**Figure 4-1** The desired file system organization to share common code.

The answer is to promote the Common Library so it is also a first-class project in Subversion: Then have both projects (Projects A and B) each

establish a link at the appropriate point in their respective project trees (Figure 4-2).

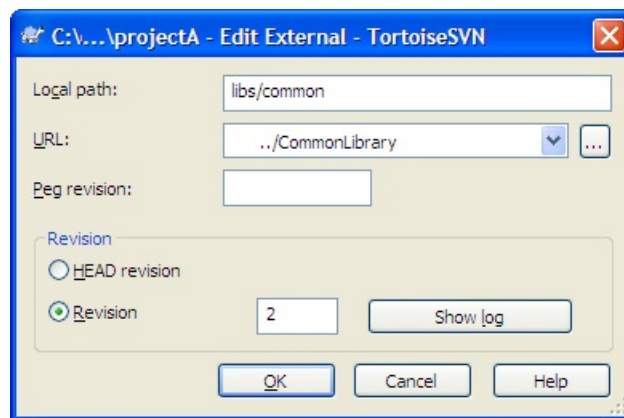


**Figure 4-2** The underlying organization of the main projects and shared project in the Subversion repository.

To set up a connection to a shared project that you want to link into your main project, you specify just three key pieces of information in the `svn:externals` property:

1. The *connection point* in your project.
2. The *address* of the target/shared project.
3. The *revision number* of the shared project.

To do this, you need to edit the `svn:externals` property. Version 1.7 of TortoiseSVN makes this easier and less error-prone by introducing a customized property editor. Figure 4-3 shows an example of the `svn:externals` property editor with the three data supplied.



**Figure 4-3** The `svn:externals` property editor.

The following sections describe the three components of the `svn:externals` definition in detail. Note, however, that you can easily experiment with the linkage without committing anything to your repository: after you define the `svn:externals` property, then perform an **SVN Update** on that item. That will pull down files from your repository if your URL is specified correctly, or yield an error if not. If you get a successful download from the repository, then examine where it places your files and fine-tune the local path appropriately if it did not place them where you expected. When you're satisfied that it's correct, commit the folder with the `svn:externals` definition.

## Connection Point (Local path)

The first key field is the connection point (the **Local path** field in Figure 4-3). *What* you specify for the connection point depends on *where* you define the `svn:externals` property. This is not some global setting but, like all Subversion properties, must be attached to one or more files or folders in your tree. This specific property, though, should be attached to just a *single* folder. There are two obvious choices for *which* folder to attach to: the project root folder or the linkage point itself. Suppose that the connection folder in Figure 4-2 for Project A is named `libs`. Underneath `libs` you want to have the `CommonLibrary` folder but with a different name, say `common`. If you choose to define `svn:externals` on the root folder of Project A, set **Local path** to `libs/common` (i.e. the path from the root to the linkage point) and include, as the leaf node of the path, the name of the folder that the linked folder will assume. If you choose instead to define `svn:externals` on the `libs` folder then just give **Local path** the value `common`. In other words, **Local path** is simply a relative file path from the folder on which you define the property to the connection point.

## Target Address (URL)

The second key field is the address of the target project (the **URL** field in Figure 4-3). You can specify the shared project's address with either an *absolute* or a *relative* path; the latter is strongly recommended in order to avoid runtime issues (e.g. protocol differences) or maintenance issues (e.g. relocated folders). Either way, be sure to properly encode the URL or it will not work. Curiously both the Subversion book, [Externals Definitions](#)

section, and the TortoiseSVN book, [External Items](#) section, mention only that the space character must be encoded as %20 in a URL. However, I suspect that *all* the standard encoding rules apply, meaning there are a variety of characters that need to be encoded—see Wikipedia’s [Percent Encoding](#) article for more.

The typical Subversion absolute URL is shown in Figure 4-4 with its component parts identified.



**Figure 4-4** An exploded view of a typical Subversion URL.

The conventional notion of a relative path, as you know, means *relative to your current directory*. Subversion, however, lets you construct a relative URL that is relative to *any one* of the four component parts identified in Figure 4-4, giving you great flexibility to select the style that fits best in your environment. You specify which style with appropriate notation.

Examining each of these in turn, first consider the nearest accessor, the conventional local (or current) directory. You know how to do this in your sleep if you have ever used a command line, but it’s a starting point! Assume that your local directory is just the Project A root folder in Figure 4-2. To reference the CommonLibrary directory, which is a sibling of Project A, you simply need to traverse “up” one to the common parent then traverse “down” to the folder of interest, i.e., `../CommonLibrary`. If you started at a folder several levels deep under Project A, use successive repetitions of the parent notation, e.g. `../../../../CommonLibrary`.

While you *could* use that last path, it is rather ugly and error-prone. So it would be quite a convenience to have a direct way to refer to your project’s repository root: Subversion uses the caret (^) notation for this. So no matter how deep you are in Project A, you can just use `^/CommonLibrary` to refer to CommonLibrary as relative to Project A’s repository root. That works because, as Figure 4-2 shows, both Project A and CommonLibrary are immediate children of the repository root. If these projects themselves are lower down, you can add additional relative path components, e.g. `^/folder1/folder2/CommonLibrary`. One important point to keep in mind with repository-relative paths: the typical Subversion repository has exactly three children—the top-level trunk, branches, and tags directories. All the projects are under the trunk subdirectory. So you would likely need to use `^/trunk/CommonLibrary` rather than `^/CommonLibrary`.

Thus far, the examples have shown how to share code from projects within the same repository. But that is not an inherent limitation of Subversion; you can easily refer to projects in different repositories and you can still use relative URLs! You might, for example, have a single parent folder containing a collection of repositories on your server, making all the repositories siblings. You can then just combine the current-directory-relative accessor with the repository-relative accessor to reference a sibling repository, as in `^/../OtherRepo/trunk/CommonLibrary`.

If, on the other hand, your repositories are not clustered together, move to the server-root-relative accessor, the virgule (/), allowing the flexibility to range across your file system with a relative URL.

Finally, be aware that if you use different schemes (or protocols) depending upon network location (e.g. `svn://` on your intranet but `svn+ssh://` on the internet), then you should use protocol-relative URLs, beginning with a double virgule (//).


The table below summarizes the relative URL accessors with both textual paths and illustrated paths. In the illustrations, the current directory is represented by the green dot and the target project is represented by the blue dot. Which type of relative URL to use will vary depending on your particular environment.

Relative to...	Path Prefix	Examples	Visualization Example
current directory	<code>..</code>	<code>../CommonLibrary</code> <code>../../../../repo/a-projects/trunk/lib</code>	
repository root	<code>^</code>	<code>^/trunk/CommonLibrary</code> <code>^/../other-repo/stuff/trunk/lib</code> <code>^/a/b/c/d/e/trunk/lib</code>	
server root	<code>/</code>	<code>/svn/my_repo/trunk/CommonLib</code>	
protocol (scheme)	<code>//</code>	<code>//example.com/svn/my_repo/trunk/CommonLib</code>	



“...relative URLs...are relative to the object where you

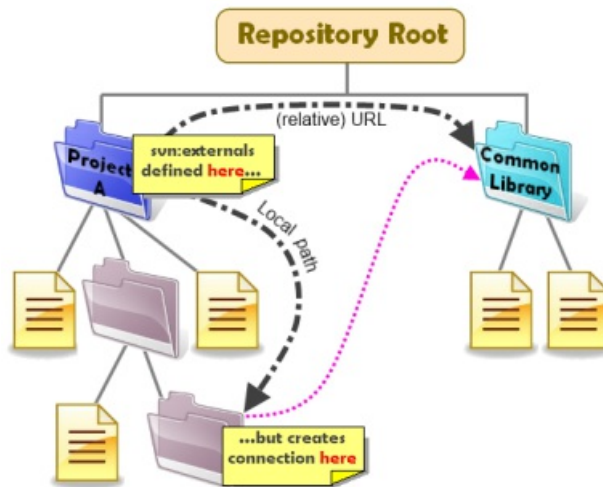
When using relative URLs, keep in mind that these are relative to the object where you define the `svn:externals` property, *not* where



define the `svn:externals` property, *not* where your `svn:externals` property specifies to actually link the external directory.”

your `svn:externals` property specifies to actually link the external directory (with the **Local path** field in the property editor). Figure 4-5 shows an example where Project A specifies an external library in the root folder of Project A itself. The heavy dashed arrows show the connection specification: the connection point in Project A (**Local path**) is two levels deeper in the Project A tree, while the

target address (**URL**) is a relative path—relative to the Project A root folder because that is where the specification is defined. The resulting connection, though, is represented by the magenta arrow.



**Figure 4-5** The specification of an external link may be far from the actual linkage point; a relative path to the target is relative to the specification point rather than the linkage point.

## Revision numbers



“You code to a specific revision of the shared library so you must explicitly define that revision in the `svn:externals` linkage.

The third key field is the revision of the target project (the **Revision** field in Figure 4-3). Note in the figure that you could select either the head revision or a specific revision. Never choose the head revision; it is a bad idea to even have it available as a choice in the dialog. Why? Think of this shared code as a single package with a revision number, just like some other single DLL or other third-party library you use in your project. You cannot just use *any* version of

`thingamabob.DLL` with your code; you code against a *specific* release. If the supplier releases an update to their DLL you do not just blindly plug it in: you review it for any API updates and breaking changes and test for runtime differences, then choose a convenient time when to make a change to a new release of the library. The situation is exactly analogous here. You code to a specific revision of the shared library so you must explicitly define that revision in the `svn:externals` linkage. The Subversion book makes the case for an explicit revision so well that the TortoiseSVN book simply repeats it:

*... you get to decide when to pull down a different snapshot of external information, and exactly which snapshot to pull. Besides avoiding the surprise of getting changes to third-party repositories that you might not have any control over, using explicit revision numbers also means that as you backdate your working copy to a previous revision, your externals definitions will also revert to the way they looked in that previous revision, which in turn means that the external working copies will be updated to match the way they looked back when your repository was at that previous revision. For software projects, this could be the difference between a successful and a failed build of an older snapshot of your complex codebase.*

In other words, you want to be in control of when your third-party code changes even if that third party code is still *your* code but from a different, shared project. The second part of that exposition is perhaps even more significant. If you ever need to go back to a previous release, *your code will still work* because you have explicit versions on your shared code. Consider the fictitious Remplissage product as an example to illustrate this. Assume you have had three releases of the product, cleverly labeled versions 1.0, 2.0, and 3.0. The table below shows the Subversion revision of your local code base associated with each label. Your product also includes a shared library that you keep in a separate repository and, per the advice given here, you specified explicit revision numbers when you linked in the shared code.

Say you have just completed and labeled Remplissage 3.0. But you have received a bug report on release 2.0, which you are naturally still supporting. So you revert to the revision associated with release 2.0 (revision 2456 from the table). Because your `svn:externals` included revision numbers your code *automatically* rolls back the shared code from revision 509 to revision 431 as well!

To my mind, it would be even better if TortoiseSVN could be redesigned to let you specify a tag name or a revision number. In development, you could easily be working with unfinished shared code so that the shared code would not be officially released, and so would not necessarily have been given a tag. However, when you are finished, or at least ready to release to production, it would be nice to be able to reference the shared code library by its tag rather than just its revision number. As it stands, you need to take the extra step to cross-reference a tag name to a revision number and then plug in the revision number in the `svn:externals` property editor.

## Bug with file scheme



As I was assembling this recipe I came across a subtle defect. In order to test scenarios, I used a local repository with the file scheme, which uses this canonical URL format: `file://host/path` (see [File URI scheme](#) on Wikipedia). You may omit the host (implicitly specifying localhost) but you must keep all the virgules, yielding three contiguous virgules: `file:///path`. The `svn:externals` property editor is actually three dialogs deep from the context menu, shown at the top of Figure 4-6. You can specify a file scheme as shown, accept the choices to close the dialog and return to the second-level dialog that now shows all of the possibly multiple values with one per line (middle of Figure 4-6). Accept the values there to return to the first-level dialog, showing all Subversion properties, one per line (bottom of the figure). That works fine the first time you add an entry using the file scheme. But I found that when I went back in to edit the `svn:externals` property—regardless of whether I touched the entry using the file scheme or not—TortoiseSVN erroneously alters the three virgules to two virgules!

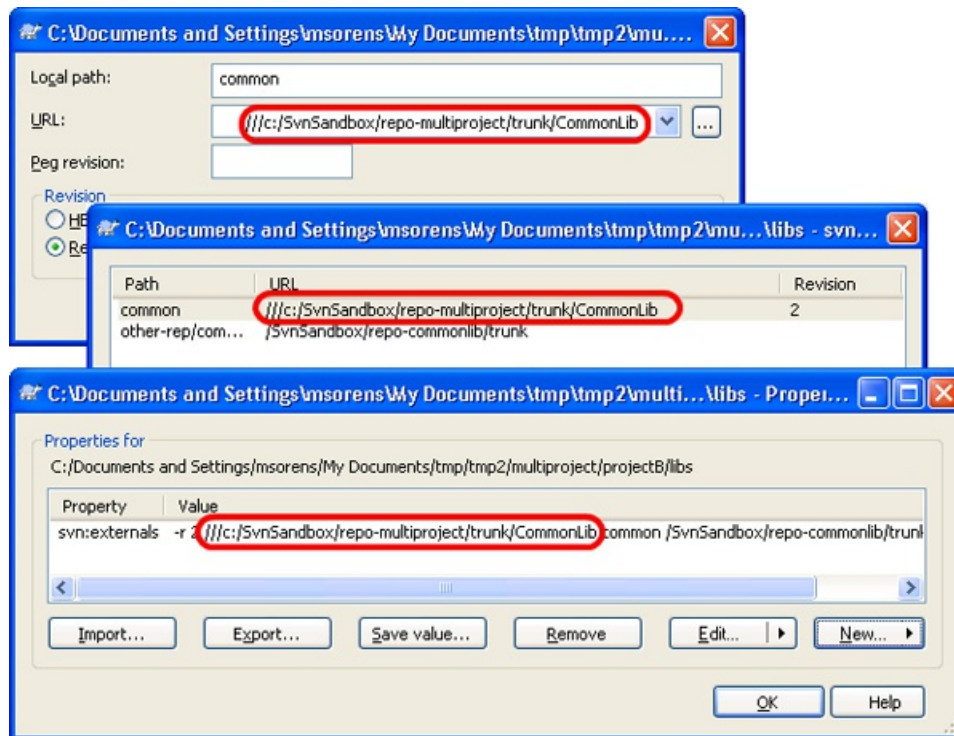


Figure 4-6 A URL using the file scheme works fine when just created, as shown here in the successive dialogs from the `svn:externals` property editor, the list of separate entries associated with the `svn:externals` property, and the list of all Subversion properties for the current object. But version TortoiseSVN 1.7.1 has a defect that changes the triple virgule to a double virgule if you re-edit the `svn:externals` property.